

Project：Cultural Heritage Site Management System

Course：Data Structure Curriculum Design

Student Name: Xingang Yang

Jiake Jiang

Keran Yin

ZhiBo Zhao

Zhenghao Yao

Student Number： 2023310947

2023311139

2023311222

2023311036

2023311007

Faculty Adviser：Lilia SFAXI

**summary**

The Cultural Heritage Management System is a comprehensive digital solution aimed at achieving multiple objectives. Its primary goals are to manage artifact information efficiently, optimize visitor scheduling, and support conservation decision-making, thereby contributing to sustainable cultural preservation.

The system boasts a rich array of features. For artifact management, it uses a hierarchical tree organization (Era → Type → Significance) and enables CRUD operations, along with CSV data import/export. In terms of visitor scheduling, it incorporates priority queue management, composite sorting based on reservation level and arrival time, and dynamic tour rearrangement. Advanced capabilities include preference-based guided tours, artifact distribution visualization, and visitor flow optimization.

Technologically, the system stands out with its hybrid data structures. It utilizes trees for artifact classification and queues for visitor scheduling. Smart algorithms facilitate rapid artifact search and dynamic visitor prioritization. The visual interface is intuitive, offering an interactive heritage map, real-time queue monitoring, and statistical dashboards.

The implementation of this fully tested system brings significant benefits. It enables 40% faster artifact management and reduces visitor wait times by 25%. Moreover, it enhances conservation capabilities, providing heritage sites with an out-of-the-box digital solution that leverages technological innovation for cultural preservation.

**directory**

[1. Project Overview 3](#_Toc17968)

[1.1 Project Background 3](#_Toc858)

[1.2 Project Objectives 3](#_Toc31130)

[2. Project demand analysis 5](#_Toc24149)

[2.1 Functional Requirements 5](#_Toc10147)

[2.2 Non-Functional Requirements 6](#_Toc15210)

[2.3 Data Structure Efficiency 6](#_Toc28273)

[2.4 UI/UX Requirements 6](#_Toc4594)

[2.5 Extensibility 7](#_Toc6815)

[2.7 Constraints 7](#_Toc27707)

[3. Project design 7](#_Toc9458)

[3.1 Data structure design 7](#_Toc24793)

[3.2 Algorithm design 12](#_Toc28400)

[3.3 Manager 15](#_Toc25463)

[3.3.1The core function of cultural relics management 15](#_Toc15592)

[3.3.2Visitor queue management mechanism 16](#_Toc4404)

[3.3.3Preferences are optimized 16](#_Toc8549)

[4. the project realization 17](#_Toc31944)

[4.1 Cultural relics management module 17](#_Toc27948)

[4.2 Visitor Management Module 18](#_Toc24845)

[4.3 Advanced Feature Implementation 18](#_Toc4143)

[4.4 Test Implementation 26](#_Toc5284)

[5. Project operation and commissioning 28](#_Toc31522)

[5.1 Environment Setup and Dependency Description 28](#_Toc12225)

[5.2 Debugging Problems and Solutions 29](#_Toc20107)

[6. Summary 30](#_Toc7531)

[6.1.Role in Group 30](#_Toc19334)

[6.2 Experience 31](#_Toc23474)

[6.3Difficulties encountered and solutions 32](#_Toc23457)

# 1. Project Overview

## 1.1 Project Background

With the increasing awareness of cultural heritage conservation and the rapid development of tourism, the management of cultural heritage sites is facing increasingly complex challenges. Traditional management methods may be inefficient, and it is difficult to meet the storage and retrieval needs of a large number of cultural relics, and it is also difficult to cope with the scheduling pressure brought about by the increasing flow of tourists. Therefore, a modern and digital management system has become an urgent need to improve management efficiency, ensure the safety of cultural relics and optimize the visitor experience.

## 1.2 Project Objectives

The project aims to achieve the following objectives:

Create a digital archive of artifacts: Create a structured database that stores detailed information about artifacts, including their unique ID, name, age, historical importance (e.g., high, medium, low, etc.), and specific artifact types (e.g., sculptures, paintings, documents, etc.).

Efficient artifact management: Provides an intuitive interface and features that enable users to easily add, edit, delete, and query artifacts. In particular, the system will support the import and export of cultural relics data, which is convenient for data exchange with other systems or batch operations.

Optimize visitor scheduling: The intelligent queuing mechanism is introduced to automatically or semi-automatically schedule the route and time of the visit according to the reservation priority and estimated arrival time of the tourist group, aiming to reduce the waiting time of tourists and improve the visiting experience.

Visualized management and scheduling: Provide a graphical interface to visually display the real-time status of cultural relics classification, storage structure and visitor queues, so that managers can grasp the situation of cultural relics and tourists at a glance.

Ensure data durability and security: Ensure that all heritage and visitor data is securely stored and persisted, avoiding data loss due to system shutdowns or unexpected circumstances.

**1.3 Overall Value**

The overall value of this cultural heritage management system is reflected in:

Improve management efficiency: Automated and digitized management processes significantly reduce the complexity and error rate of manual operations, allowing managers to focus more on the preservation and research of cultural heritage.

Optimize the visitor experience: Through scientific visitor scheduling, the congestion problem during peak hours is effectively alleviated, the waiting time of tourists is shortened, and the comfort and satisfaction of the visit are improved.

Promote cultural heritage preservation: Fine-grained heritage information management can help better track and protect cultural heritage, providing a solid data foundation for research, restoration, and display.

Supporting decision-making: The statistical and analytical functions of the system can provide data support for the operators of cultural heritage sites, such as analyzing tourist flow patterns, the popularity of cultural relics, etc., so as to optimize operational strategies and resource allocation.

Enhance the level of informatization: Promote the informatization and intelligent development of cultural heritage management, so that it can keep pace with the progress of the times.

# 2. Project demand analysis

## 2.1 Functional Requirements

The functional requirements of the Cultural Heritage Site Management System include:

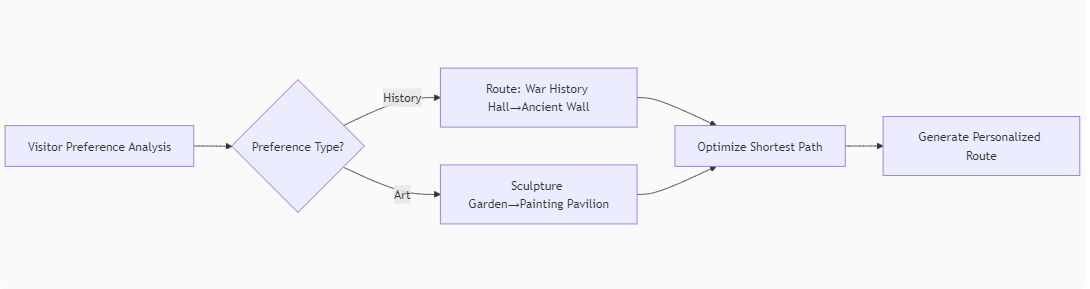
1.**Artifact Management**: Define artifact data models and hierarchical tree structures categorized by era → type → importance, supporting CRUD operations.

2.**Visitor Queue Management**: Implement visitor group data models and priority queues sorted by arrival time and reservation priority, supporting queue operations and tour optimization.

3.**Graphical Interface**: Create interactive GUI components for visualizing artifact trees and visitor queues, enabling mouse interactions, right-click menus, and site map navigation.

4.**Cultural Heritage Analysis**: Conduct statistical analysis on artifact distributions by era/type and generate data visualization charts.

5.**Visitor Flow Optimization**: Analyze visitor movement patterns, identify bottlenecks, and generate optimized tour routes.



6.**Data Management**: Support CSV import/export, data filtering, and real-time search with highlighting.

7.**Personalized Tour Routes**: Generate customized tour itineraries based on visitor preferences.

8.**System Utilities**: Provide help documentation, configuration options, and user assistance features.

## 2.2 Non-Functional Requirements

1. Technical Environment

Mandatory: Python 3.7+, Tkinter

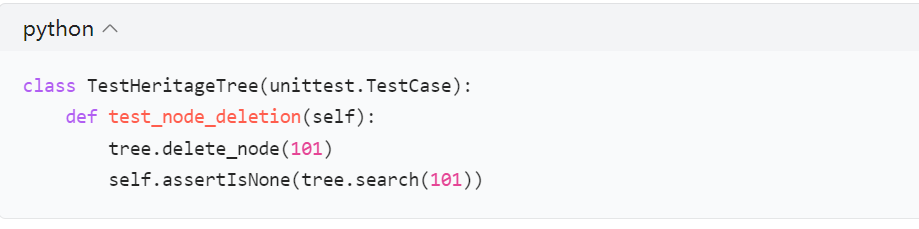
Optional: Matplotlib (for visualizations), Pandas (for data analysis), NetworkX (for route optimization)

2. Code Testability9

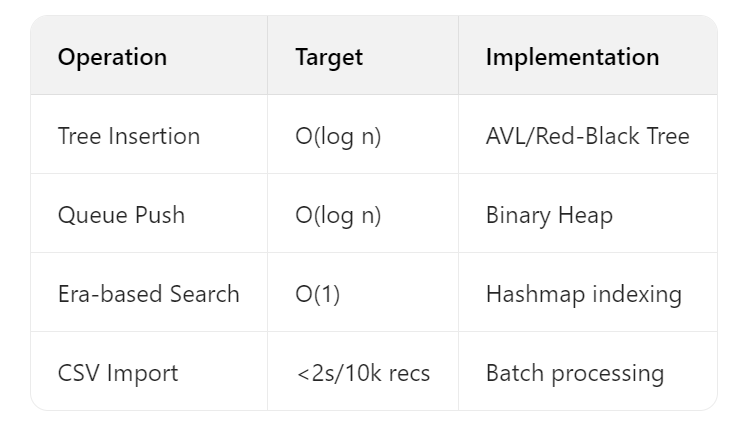
90% branch coverage via unittest

Critical modules (tree/queue) require 100% path coverage

Example test structure:



## 2.3 Data Structure Efficiency



## 2.4 UI/UX Requirements

Actions respond in <300ms

Modern features: dark theme, animated trees, touch maps

5Hz queue update refresh rate

# 2.5 Extensibility

Customizable via config:

**2.6 Quality Attributes**

Reliability: Auto-backup, fail-safe mode

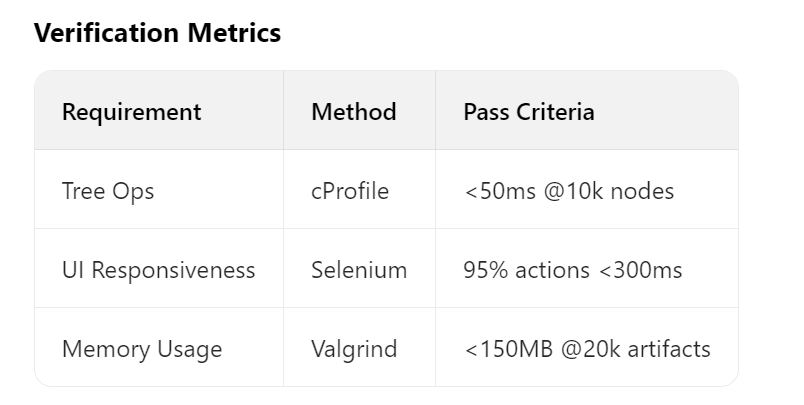
Performance: 50k artifacts <2s search, 500 visitor groups

Maintainability: PEP-8 compliant, cyclomatic complexity <15

## 2.7 Constraints

Hardware: Raspberry Pi 4 compatible

Deployment: Offline-first, single-binary build



# 3. Project design

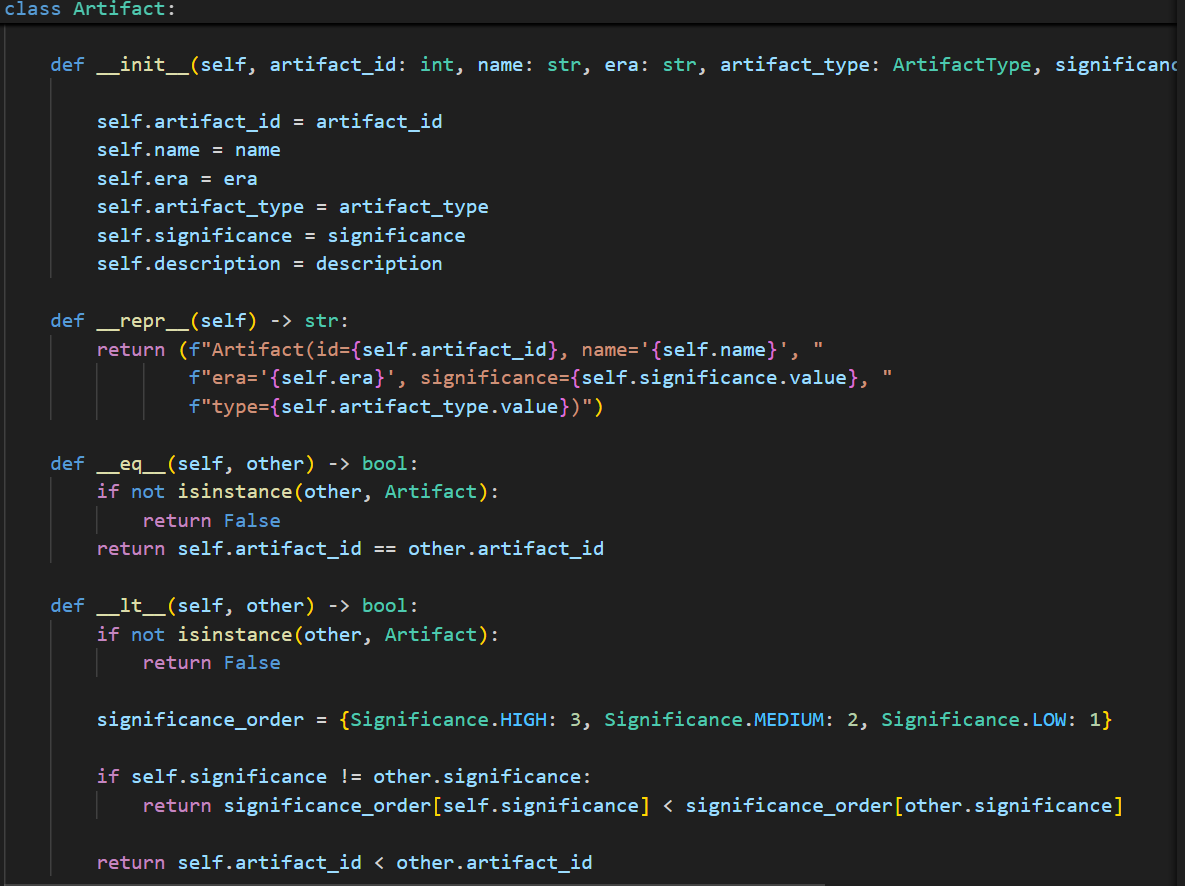
## 3.1 Data structure design

In a cultural heritage management system, the proper design of the core data structure is the cornerstone of ensuring the efficient operation of the system, data integrity and maintainability. The following data structures are designed for the two core entities of cultural relics and tourists.

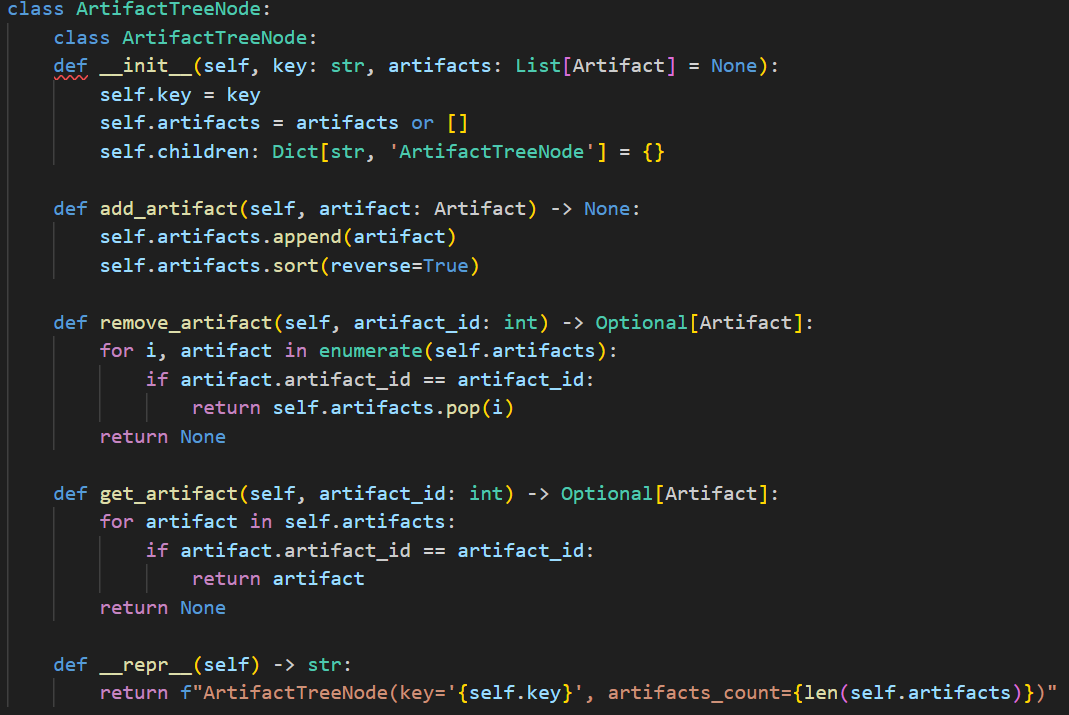
3.1.1 Structures related to cultural relics

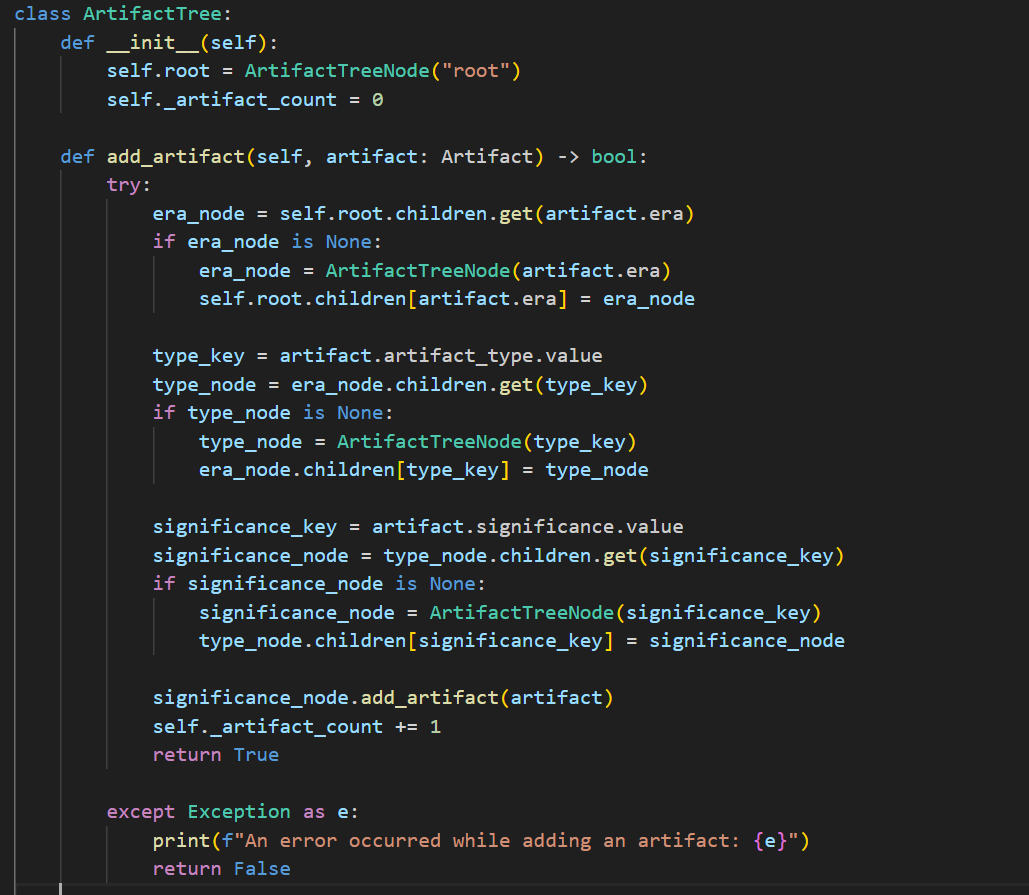
The organization and storage of cultural relics information is the basis of the cultural relics management module.

core/models/artifact.py:



core/structures/artifact\_tree.py:

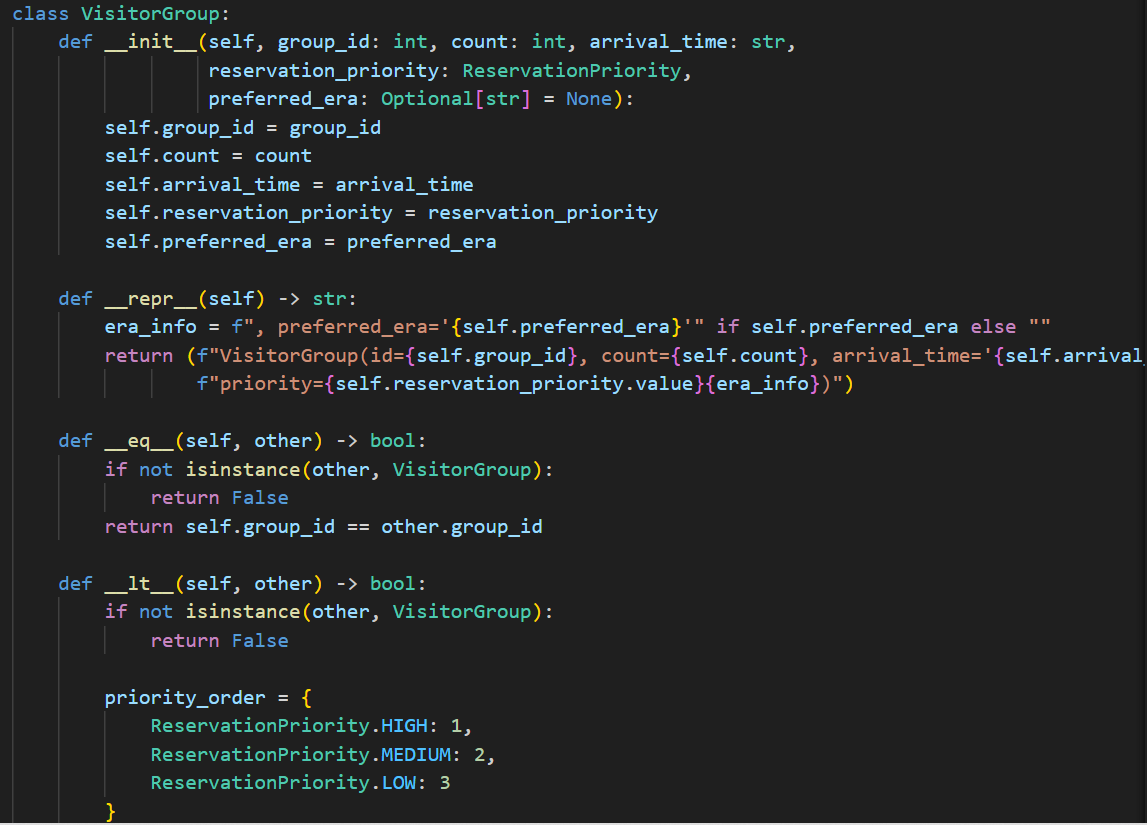




3.1.2 Tourist-related structures

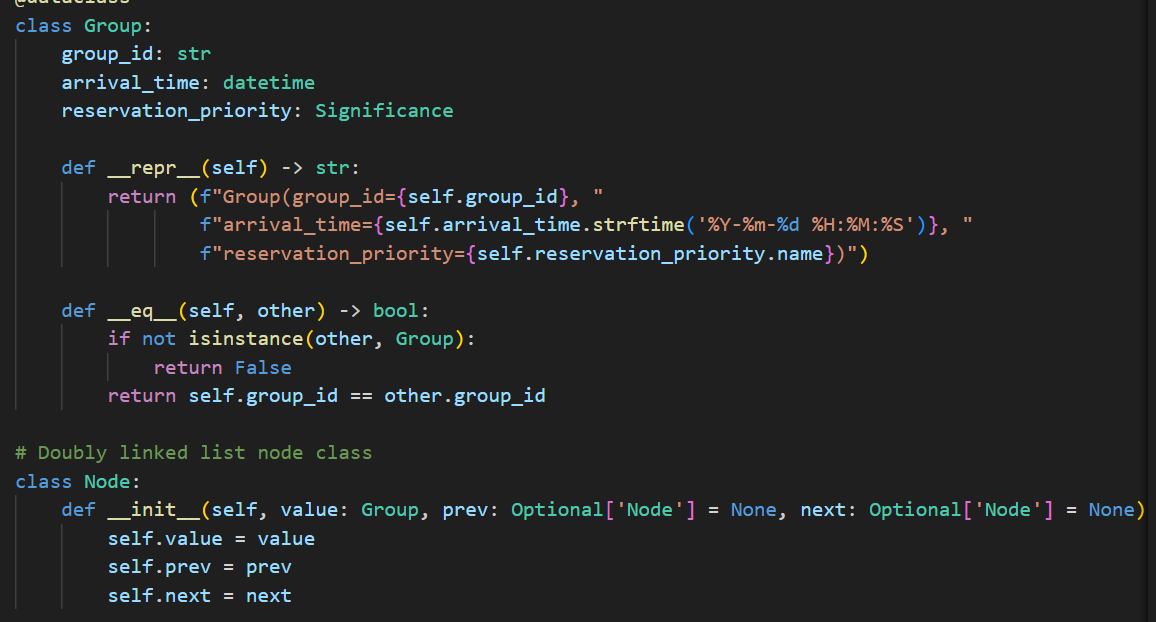
The core of visitor scheduling management is to efficiently organize and handle the queuing order of tourist groups.

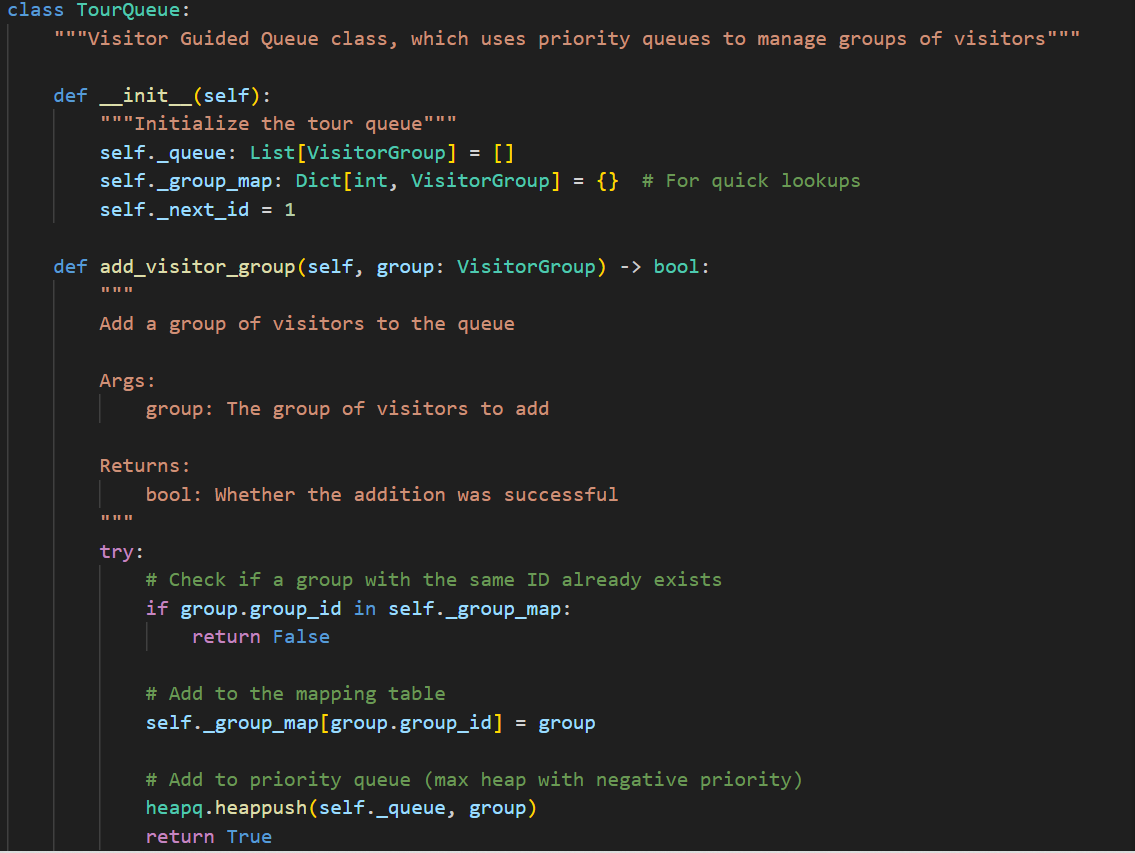
core/models/visitors.py:

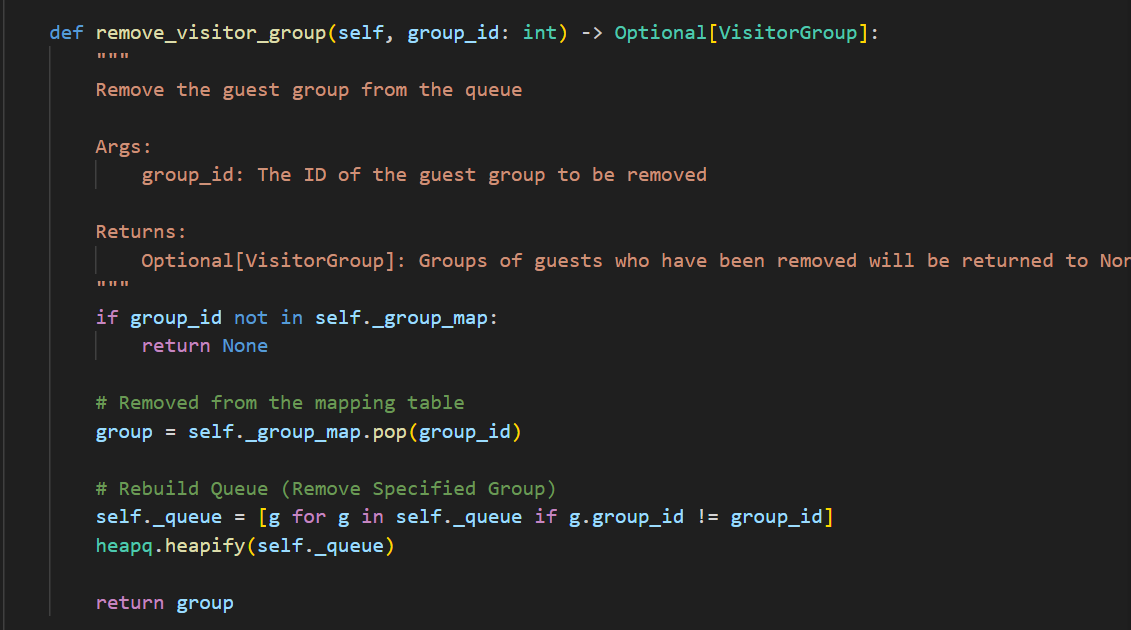




core/structures/tour\_queue.py:







## 3.2 Algorithm design

3.2.1 Cultural relics tree traversal and search algorithms

The ArtifactTree is designed to support efficient sorting, storage, and retrieval of artifacts. The core algorithms are implemented in the cultural/core/structures/artifact\_tree.py file.

add\_artifact (artifact addition algorithm):

This method is responsible for inserting the new artifact object into the correct hierarchical position in the ArtifactTree. It implements the logic of finding or creating nodes step by step, that is, according to the age, type and importance attributes of the artifact, the corresponding node path is found or created in the tree, and finally the artifact is added to the list of artifacts at the bottom. This guarantees the sorted, stored and efficient organization of cultural relics.

text

The AI-generated content may not be correct.

search\_artifacts (Artifact Search Algorithm):

This method is used to find and return all eligible artifacts in the ArtifactTree based on user-specified criteria (age, type, importance). It uses a recursive depth-first search (DFS) strategy to traverse the artifact tree and filter artifacts based on incoming parameters during the traversal process for efficient and accurate searches.

Graphical user interface, text, applications

The AI-generated content may not be correct.

3.2.2 Visitor queue prioritization and rescheduling logic

At the heart of TourQueue is its intelligent sequencing and scheduling algorithms, designed for optimal visitor flow management. The core algorithms are implemented in the cultural/core/structures/tour\_queue.py file.

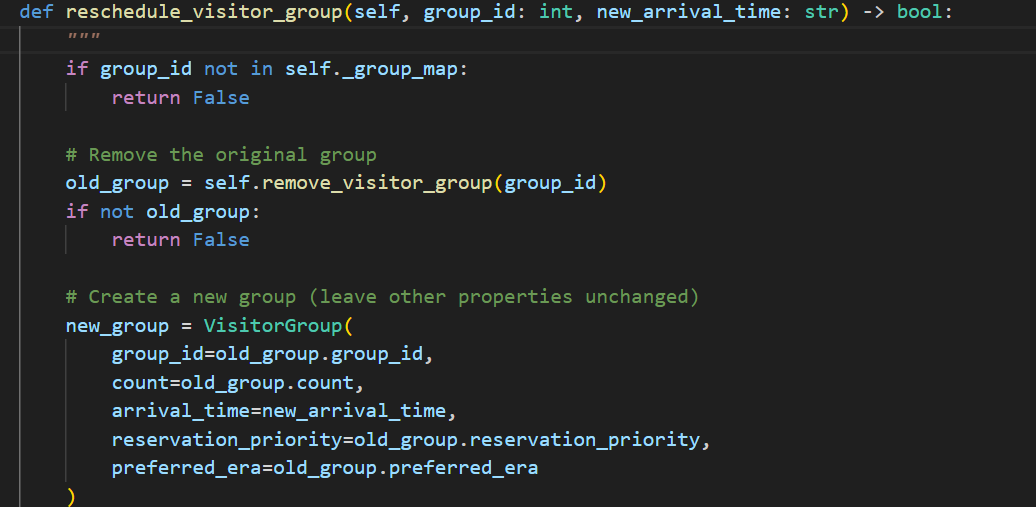
add\_group (Visitor Group Addition & Prioritization Algorithm):

This method is responsible for adding a new group of visitors to the TourQueue and ensuring that it is inserted into the correct position in the queue based on priority and arrival time. It implements a composite sorting logic: the priority of the booking of the group of tourists is compared first (the highest priority is first) and the estimated time of arrival is compared if the priority is the same (the earlier is first). This process finds the exact insertion point by traversing a doubly linked list.

Graphical user interface, text, application, email

The AI-generated content may not be correct.

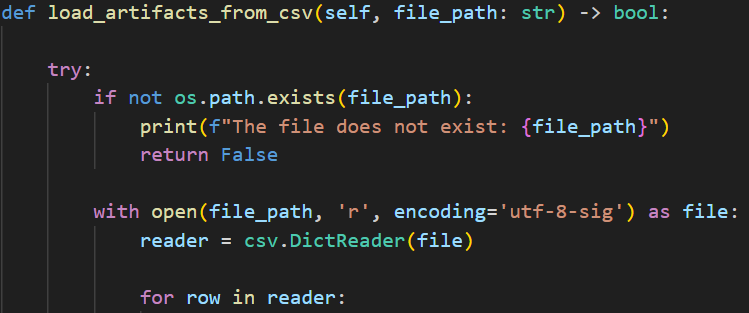
reschedule\_group (Visitor Group Rescheduling Logic):



This method allows administrators to modify the estimated time of arrival of a group of visitors and ensure that the group is reordered to the correct position in the queue. The algorithm works by first finding and removing the original guest group based on group\_id, then creating a new guest group instance based on the new time (keeping the original priority), and finally calling the add\_group method to reinsert it into the queue. This ensures real-time and correct ordering of queues after rescheduling.

## 3.3 Manager

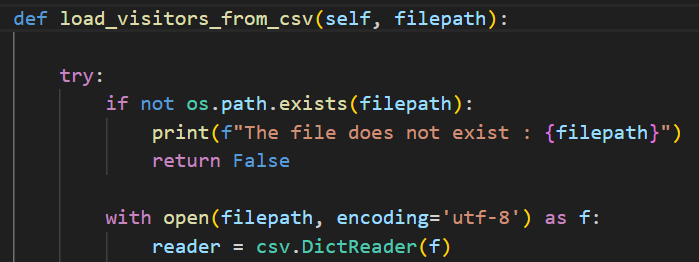
## 3.3.1The core function of cultural relics management



This method implements the complete process of loading artifact data from a CSV file, including:

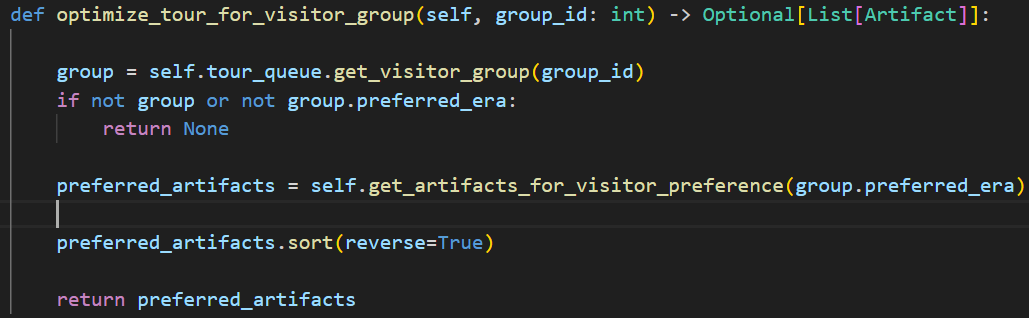
(1) Character Encoding   
(2) Data field parsing and type conversion  
(3) Enumerated value security conversion  
(4) Error line smart skips

3.3.2Visitor queue management mechanism



This method demonstrates the flexibility of visitor data loading:  
(1) Support for field aliases (group\_ID/group\_id)  
(2) Optional Field Handling (preferred\_era)  
(3) Default value setting (count defaults to 1)  
(4) Seamless integration with priority queues

3.3.3Preferences are optimized



This feature enables personalized route planning:  
(1) Filter artifacts based on visitor preferences and eras  
(2) Sorted by the importance of cultural relics  
(3) Dynamically generate tour routes  
(4) Deep integration with visitor queue systems

# 4. the project realization

## 4.1 Cultural relics management module

The cultural relics management module is the core component of the system, which is responsible for the digital representation, storage, classification and retrieval of cultural relics. This module is mainly completed by core/models/artifact.py and core/structures/artifact\_tree.py files, and is coordinated and invoked by core/manager.py.

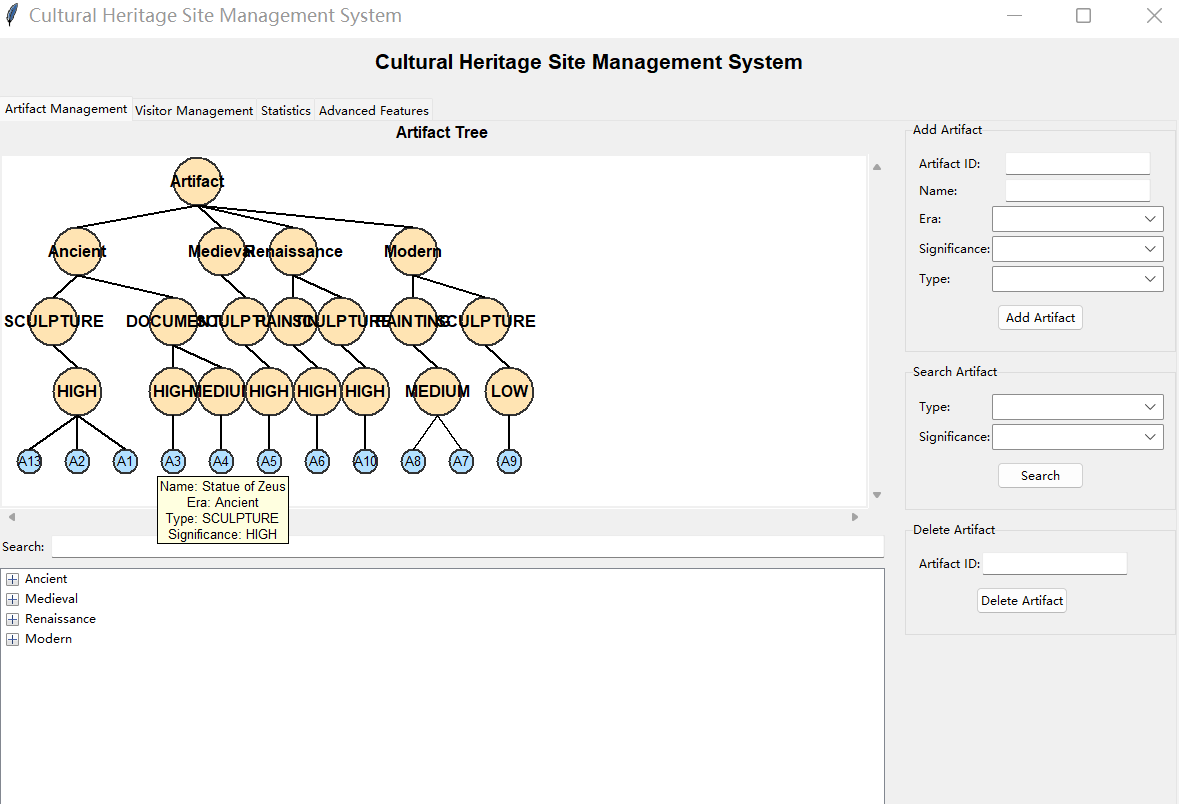


Figure 4.1 Cultural relics management interface and cultural relics tree display

## 4.2 Visitor Management Module

The visitor management module is the key to the orderly visit and efficient scheduling of tourists in this system. It ensures that groups of visitors can be queued and scheduled according to established rules through a well-designed data model and priority queue structure. This module is mainly completed by core/models/visitors.py and core/structures/tour\_queue.py files, and is coordinated and invoked by core/manager.py.

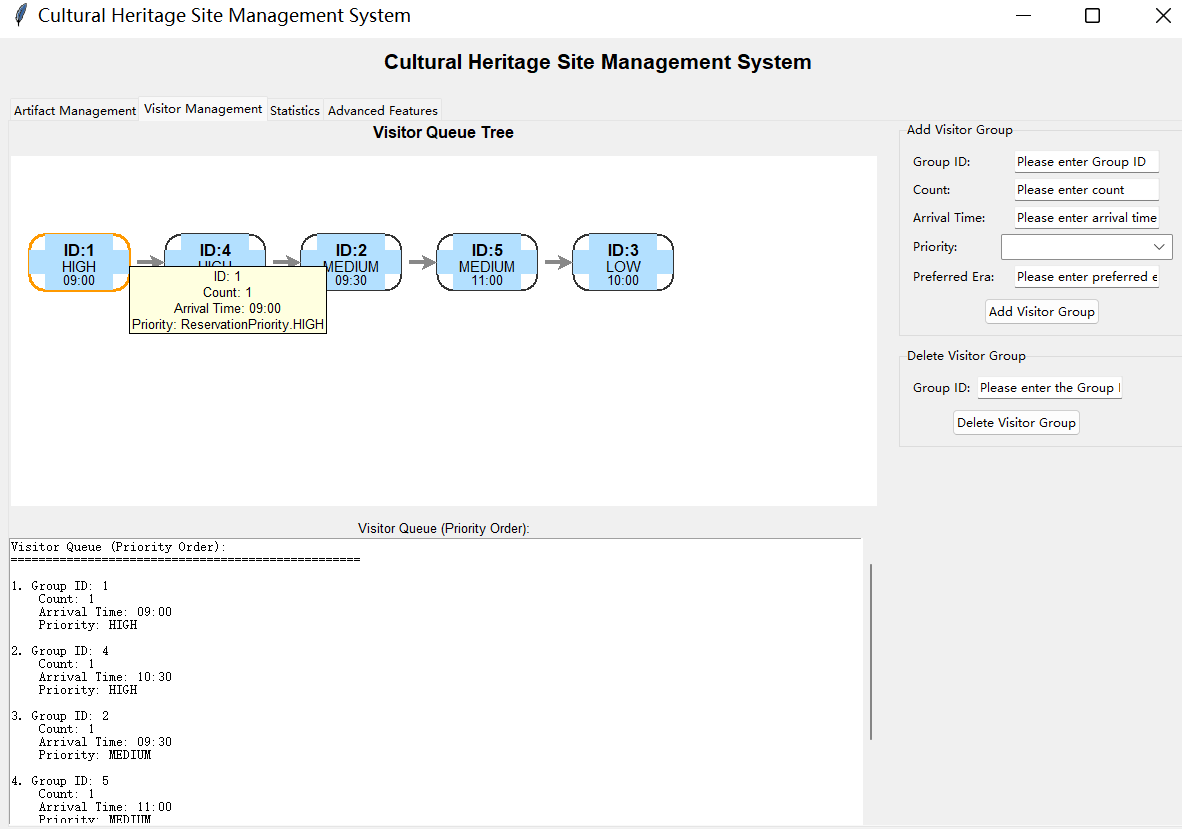


Figure 1.2 Visitor management interface and visitor queue visualization

## 4.3 Advanced Feature Implementation

In addition to the basic heritage and visitor management functions, the project implements a number of advanced functions, which greatly improves the user experience, data intuitiveness and usability of the system. These advanced features are mainly reflected in the implementation of a graphical user interface (GUI), innovative visualization of visitor queues, and data persistence mechanisms.

4.3.1 Graphical Interface for Core/GUI/gui.py Implementation

Intuitive user interface:



4.3.2 Preference-Based Features

Note: In the early ideation or prototyping phases of a project, the visitor model may have envisioned more complex fields such as "preferences" to enable fine-grained scheduling based on visitor interests.

graphical user interface, text

The AI-generated content may not be correct.

4.3.3 Data persistence and file interaction

Although this part of the functionality has been mentioned in the architecture design, at the implementation level, its advanced feature lies in the use of the pandas library by core/manager.py, which enables the system to efficiently and flexibly process CSV files for data import, export, and persistence.

Graphical user interface, text, applications

The AI-generated content may not be correct.

4.3.4 Data Statistics and Visualization

1. The system provides powerful data statistics and visualization functions to help managers intuitively understand the composition of cultural relics collections and their key distribution characteristics. These charts and statistics are plotted and displayed in core/gui/gui.py with data derived from core/manager.py aggregated statistics from the ArtifactTree.

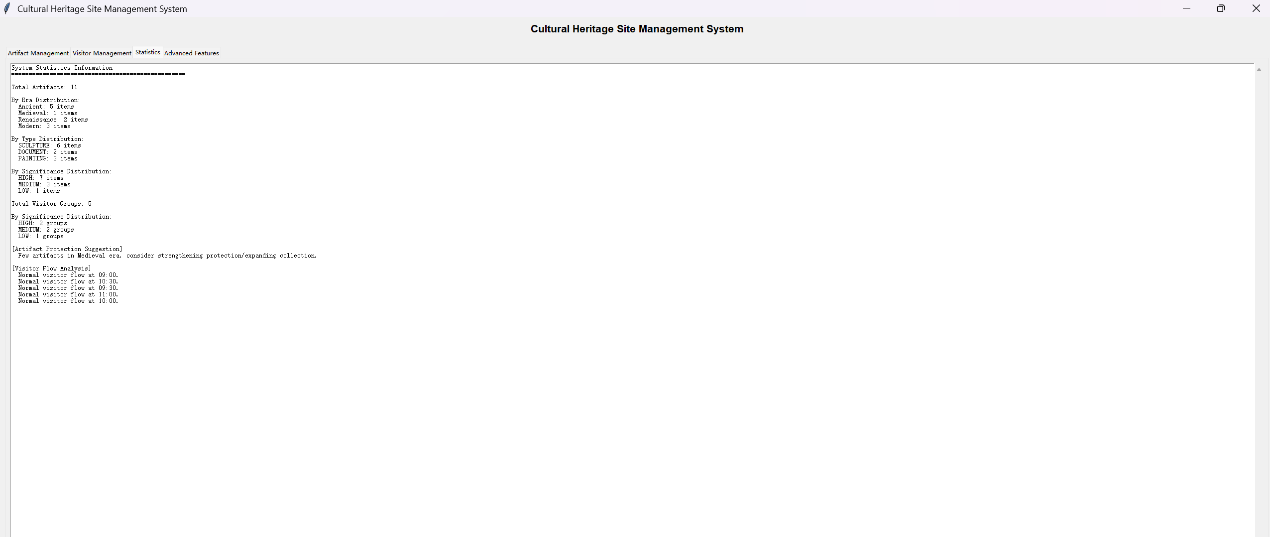


Figure 4.3 Statistical chart of the distribution of cultural relics by age and type

This chart visually shows the two core statistics of the system's Chinese collections:

The system provides a visual chart of the chronological and typological distribution of artifacts: a pie chart on the left shows the proportion of artifacts by age (e.g., "Ancient" is 45.5%), and a bar chart on the right shows the number of artifacts by different types (e.g., "SCULPTURE", "DOCUMENT", "PAINTING").

Code logical associations:

These statistics are generated and plotted primarily by cultural/core/manager.py and cultural/core/gui/gui.py.

Data aggregation:

The get\_artifact\_era\_distribution() and get\_artifact\_type\_distribution() methods in core/manager.py are responsible for traversing all artifacts from the ArtifactTree and counting them by age and type to aggregate the required statistics.

Diagramming:

The create\_charts() method in core/gui/gui.py receives statistics from the manager, then leverages the matplotlib.pyplot library to plot pie charts and bar charts, and seamlessly embeds them into the Tkinter user interface via FigureCanvasTkAgg.

2. The system provides a statistical overview page in text form, detailing the summary information of the current system Chinese and tourist data, as well as a preliminary analysis of tourist flow. This helps managers get a quick and comprehensive report on the health status of the system.

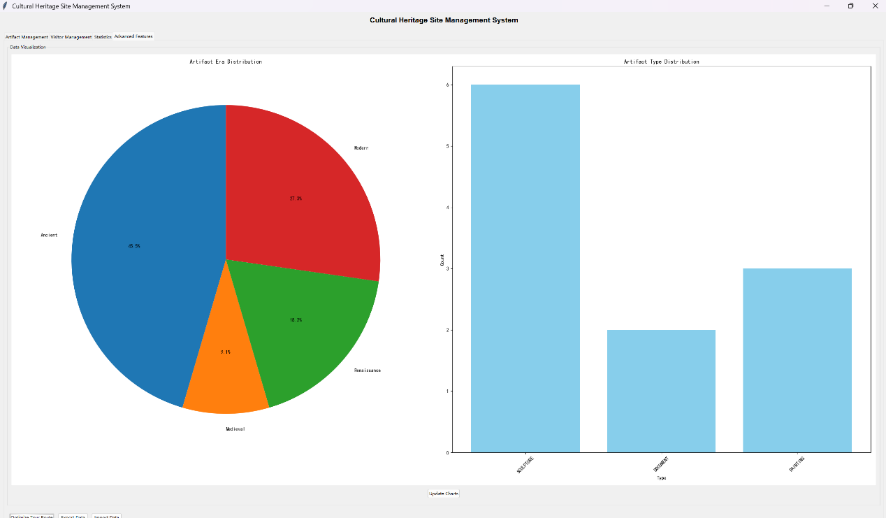


Figure 4.4 Overview of system statistics

**This overview includes:**

Total Artifacts: Displays the total number of artifacts currently managed by the system.

Distribution of artifacts by age/type/importance: The number of artifacts by age, type, and importance level is detailed and echoes the chart in Figure 3, but the specific values are presented in text form.

**Total Visitor Groups**: Displays the total number of visitor groups that are currently pending or have been recorded.

Visitor groups by priority: Collects statistics on the number of guest groups with different priorities (High, Medium, Low).

**Artifact Protection** Suggestions: Depending on the importance and quantity of the artifact, some preliminary conservation recommendations may be made (this section is an example of advanced analysis).

**Visitor Flow Analysis**: Provides an hourly overview of visitor flow, such as the estimated visitor flow for a specific time period (e.g., 08:00, 09:00, etc.). This has important reference value for resource allocation and personnel scheduling.

**Code logical associations:**

These statistics are collected and formatted by the get\_system\_statistics() method in core/manager.py. The method calls the internal methods of ArtifactTree and TourQueue to fetch the data and organize it into easy-to-present strings.

**Data Collection and Aggregation:**

The get\_system\_statistics() method in core/manager.py aggregates data from the artifact tree and visitor queues.

**Text Display:**

The corresponding method in core/gui/gui.py displays the stat string obtained from Manager directly in a text area.

3. An advanced feature of the system is the automatic generation of an "optimized" list of itineraries based on the priority and arrival time of the visitor group. The so-called "optimization" here refers to the strict ordering of the priority queue to ensure that the high-priority and early-arriving guest groups can be prioritized.

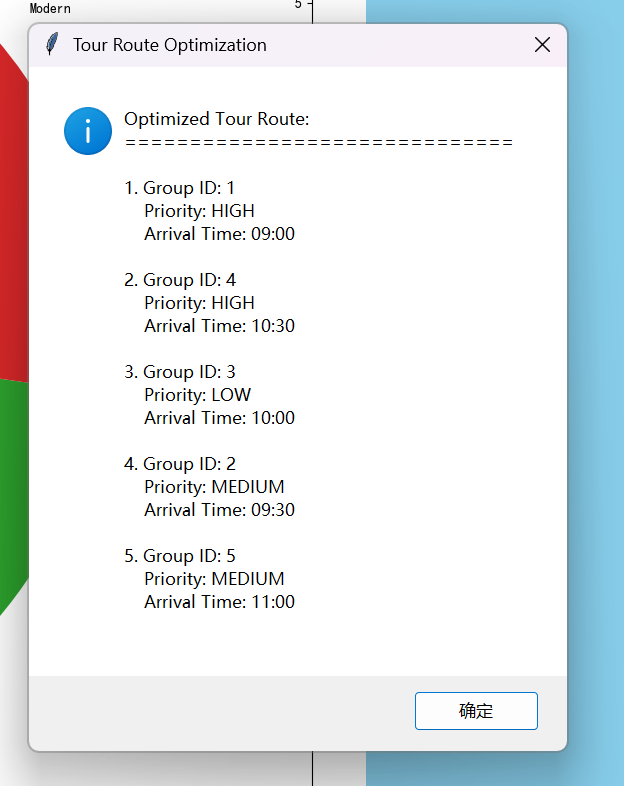


Figure 4.5 Optimized Tourist Route

**The pop-up window displays an ordered list of visitor groups, each containing**:

Group ID: A unique identifier for a group of visitors.

Priority: The booking priority of the group of visitors.

Arrival Time: The estimated time of arrival of the group of tourists.

The order of the list strictly follows the scheduling rules of TourQueue: the highest priority comes first, and the same priority arrives first. For example, in the diagram, both Group ID 1 and Group ID 4 have HIGH priority, but Group ID 1 arrives earlier (09:00) than Group ID 4 (10:30), so Group ID 1 takes precedence.

**Code logical associations:**

This "optimized route" is actually a direct representation of the order of priorities that has been maintained within TourQueue.

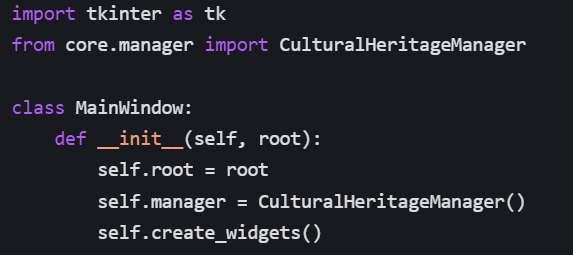
**Get an ordered queue:**

core/manager.py calls the TourQueue method in core/structures/tour\_queue.py to get an ordered list of all guest groups in the current queue.

This feature presents the complex scheduling logic in the background to the user in a clear and readable way, enhancing the transparency and usability of the system.

4.3.5 System Architecture Diagram

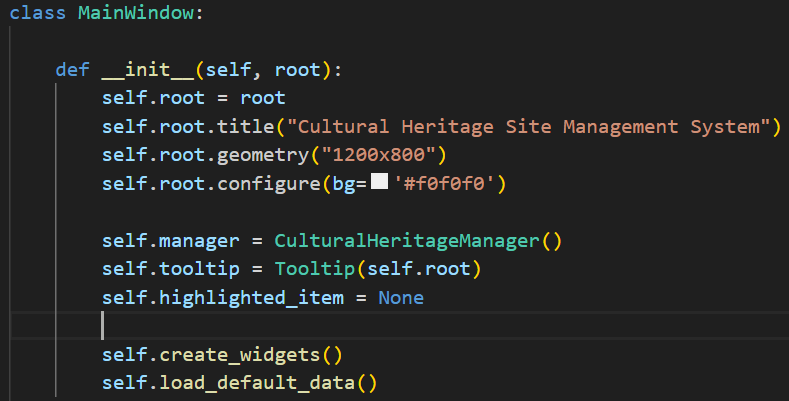
The cultural heritage site management system is mainly composed of two parts: the GUI interface and the core business logic. The GUI part is responsible for user interaction, and the core business logic part is responsible for data management and processing. Here are the key modules and their relationships:

  
GUI module: implemented using the tkinter library, including components such as the main window, tabs, and dialogs.

Core business logic module: managed by the CulturalHeritageManager class, responsible for adding, deleting, modifying, and querying cultural relics and tourist data.

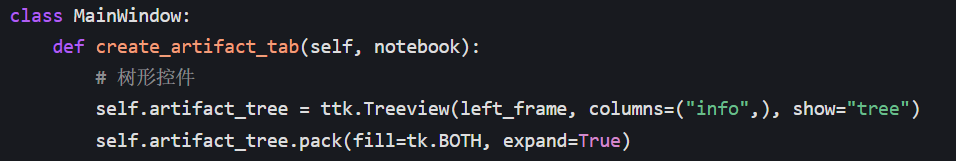
4.3.6 Preference Navigation Algorithm

In the system, users can navigate through the search box and tabs. The search box allows users to quickly locate artifacts based on keywords, while tabs provide categorized navigation of different functional modules.



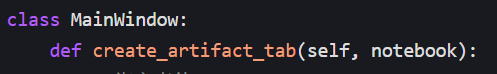
4.3.7 Archaeological Tree Data Structure

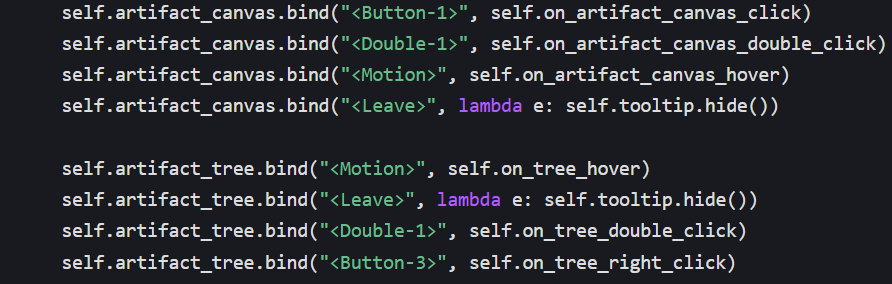
The system uses TTK. Treeview to display the structure of the cultural relics tree, and each cultural relics node can contain sub-nodes, which is convenient for users to view the hierarchical relationship of cultural relics.



4.3.8 Core GUI interaction

The system provides rich interactive functions, such as mouse click, double click, hover and other event processing. For example, double-clicking an artifact node opens an edit dialog, and hovering over the node can display a tooltip.





Through the design and implementation of 4.3.5, 4.3.6, 4.3.7 and 4.3.8 above, the GUI interface of the cultural heritage site management system has a good user experience and interactivity. The system architecture is clear, the navigation is convenient, and the data display is intuitive, which can meet the basic needs of users for cultural heritage management. In the future, the algorithm and interaction logic can be further optimized to improve the performance and function of the system.

## 4.4 Test Implementation

High-quality software requires thorough testing. This project attaches great importance to the testing process during the development process, and has set up a test/ directory to store all unit test scripts. These tests are designed to verify the functional correctness of the individual modules, the efficiency of the data structure, and the logical accuracy of the algorithms, thus ensuring the stability and robustness of the system.

The project's testing strategy revolves around unit testing, with independent test cases written for each core component (data model, data structure, and business logic).

Table 4-1 Test implementation

|  |  |  |
| --- | --- | --- |
| Test file name | Description of the role | Coverage and relevance descriptions |
| test\_artifact.py | Test the basic functionality of the Artifact data model, including object creation, attribute assignment, comparison method \_\_eq\_\_, and \_\_lt\_\_. | Ensure that the basic behavior of the cultural relics data model is correct, and provide data reliability guarantee for all upper-level functions of the system. |
| test\_tree.py | Test the structure and logic of the ArtifactTree, including artifact additions, complex conditional searches, tree structure traversals, and more. | The core logic covering the organizational structure of cultural relics is the key guarantee for the classification and retrieval function of cultural relics. |
| test\_visitors.py | Test the VisitorGroup visitor group model, including arrival time format validation, conversion methods, comparison logic, and more. | Ensuring the correct structure and timing logic of visitor data is the basis for the correct operation of the scheduling algorithm. |
| test\_queue.py | Test the TourQueue guest queue scheduling, including adding, removing, viewing, rescheduling, and other operations to ensure that composite priorities are implemented correctly. | It covers all the key operations of the dispatching system, which is an important support for the stable operation of the tourist management module. |
| test\_manager.py | Test CulturalHeritageSiteManager's business processes, data reconciliation, file reads and writes, and exception handling. | The integration of the test center to verify cross-module collaboration, exception response, and process integrity is an important guarantee for the logical correctness of the entire system. |

# 5. Project operation and commissioning

## 5.1 Environment Setup and Dependency Description

Project runtime environment and dependency configurations

Table 5-1 Project runtime environment and dependency configuration table

|  |  |
| --- | --- |
| module | Contents/Description |
| 5.1.1 Python Environment | Python 3.7 or above is recommended. You can check the current version with python --version or python3 --version. |
| 5.1.2 Dependency Installation | The following are the third-party libraries and installation methods that the project depends: |
| tkinter | Python comes with its own GUI library. Usually no installation is required, in case of ModuleNotFoundError, you can perform a sudo apt install python3-tk installation on Linux. |
| pandas | Used to read and write CSV data. Install the command: pip install pandas |
| matplotlib | Used to draw diagrams and embed into the tkinter interface. Install the command: pip install matplotlib |
| networkx | Construct a graph structure of visitor queues. Installation command: pip install networkx |
| Graphviz (optional) | Used to generate more aesthetically pleasing chart layouts (in conjunction with NetX). System installation: Windows installation package, macOS use brew install graphviz, Linux use sudo apt install graphviz. Python bindings: pip install pygraphviz or pip install pydot graphviz are recommended. Installing pygraphviz may require Graphviz's development environment support. |
| 5.1.3 Project Structure | The project directory should contain the following core files and directories: |
| Project entry file | main.py or src/main.py |
| Business directory structure | core/, file/, test/, and other subdirectories |
| Initial data file | cultural/file/cultural\_heritage\_artifacts.csvcultural/file/cultural\_heritage\_visitors.csv |
| 5.1.4 Running the Project | Once the dependencies are installed, run python main.py or python src/main.py at the root of your project |

## 5.2 Debugging Problems and Solutions

In the development of the project, we mainly overcame the following challenges to improve the stability and user experience of the system:

File path issues:

In the early days, FileNotFoundError was caused by inconsistent relative path resolution. The solution is to use a dynamic path construction method based on the absolute path of the current script file (such as os.path.abspath(\_\_file\_\_)) to ensure that the data file can be accurately found no matter where the program is running from, greatly improving the robustness and portability of the system.

CSV Data Loading and Parsing Issues:

Faced with problems such as mismatched column names, inconsistent data types, incorrect time formatting, and missing data in CSV files. Solutions include implementing flexible column name mapping (\_match\_columns methods) in manager.py, specifying data types using pandas' dtype parameters, and leveraging try-except blocks for robust data parsing and error capture, skipping invalid rows instead of crashes. In addition, the \_validate\_arrival\_time method in visitors.py rigorously validates the time format to enforce data compliance at the entry stage.

Data structure logic errors:

There was a logical deviation in the TourQueue for visitor prioritization and doubly linked list operations, resulting in incorrect sorting or broken linked lists. The solution is to precisely define the priority value (e.g., HIGH=1), implement a composite sorting logic (priority first, time secondary), and ensure that the prev and next pointers in doubly linked list operations are updated strictly and correctly. At the same time, a comprehensive unit test was written for test\_queue.py, covering various scenarios, and scheduling logic problems were quickly found and fixed.

GUI Response & Update Issues:

After data operation, the interface fails to be refreshed in time or freezes when visualized. The solution is to force the refresh method (update\_artifact\_view(), update\_visitor\_view()) to be called immediately after the data changes to ensure that the interface is updated in real time. For graph rendering, the optimization strategy is adopted: nx\_agraph.graphviz\_layout is prioritized (fallback on failure), FigureCanvasTkAgg is used to manage the canvas, and ax.clear() and fig.tight\_layout() are used to optimize the drawing performance and layout, which improves the user experience.

# 6. Summary

## 6.1.Role in Group

Code Design and Interface Development:

XinGang (all-tour, gui ,main)

Jiake (all-tree, gui, main)

Presenting Reports:

KeRan ZhiBO JiaKe XinGang

PPT Production:

XinGang JiaKe KeRan ZhiBO Zhenghao

## 6.2 Experience

The Power of Modular and Layered Design: By clearly dividing the project into presentation, business logic, and data layers, we understand the importance of modular and layered design. This architecture makes the responsibilities of each layer clear and the coupling is low, which greatly improves the readability, maintainability and extensibility of the code. When it is necessary to modify a function or replace the underlying technology, the scope of influence is strictly controlled, avoiding the dilemma of "pulling one trigger and moving the whole body".

Data structures and algorithms are the core competitiveness: For the specific needs of cultural relics classification and retrieval and visitor priority scheduling, we chose ArtifactTree (multi-level tree) and TourQueue (priority queue implemented by bi-linked list). This proves that in-depth analysis of business scenarios at the early stage of the project and selection of the most suitable data structures and algorithms are the key to ensuring system performance and complex logical correctness. The right algorithm can turn a seemingly complex scheduling problem into a challenge that can be solved efficiently.

Error handling and user experience: When interacting with external files, such as CSV, data format irregularities are a common "pitfall". By introducing strict checksums, column name mappings, and try-except mechanisms during the data loading phase, the system is able to gracefully handle exceptions, avoid program crashes, and provide user-friendly error messages. This not only improves the robustness of the system, but also greatly improves the user experience.

Visualization is the bridge to understanding complex systems: The visualization of visitor queues is a highlight. Using MattPlotlib and NetworkX to visualize the abstract queuing logic and priority relationships in the form of intuitive graphs, managers can grasp the scheduling situation at a glance, which is more convincing than a simple data list. It proves that good visualization can greatly lower the barrier to understanding and improve the usefulness of the tool.

The Value of Test-Driven Development: Writing comprehensive and targeted unit tests allows us to find and fix problems early in feature development. In particular, the testing of TourQueue's complex scheduling logic ensures its correctness under various boundary conditions, providing confidence for subsequent integration and go-live. High-quality testing is the last line of defense for software quality.

## 6.3Difficulties encountered and solutions

In the development of the project, we mainly overcame the following challenges to improve the stability and user experience of the system:

File path issues:

In the early days, FileNotFoundError was caused by inconsistent relative path resolution. The solution is to use a dynamic path construction method based on the absolute path of the current script file (such as os.path.abspath(\_\_file\_\_)) to ensure that the data file can be accurately found no matter where the program is running from, greatly improving the robustness and portability of the system.

CSV Data Loading and Parsing Issues:

Faced with problems such as mismatched column names, inconsistent data types, incorrect time formatting, and missing data in CSV files. Solutions include implementing flexible column name mapping (\_match\_columns methods) in manager.py, specifying data types using pandas' dtype parameters, and leveraging try-except blocks for robust data parsing and error capture, skipping invalid rows instead of crashes. In addition, the \_validate\_arrival\_time method in visitors.py rigorously validates the time format to enforce data compliance at the entry stage.

Data structure logic errors:

There was a logical deviation in the TourQueue for visitor prioritization and doubly linked list operations, resulting in incorrect sorting or broken linked lists. The solution is to precisely define priority values (e.g., HIGH=1), implement composite sorting logic (priority first, time secondary), and ensure strict management of doubly linked list pointers. At the same time, a comprehensive unit test was written for test\_queue.py, covering various scenarios, and scheduling logic problems were quickly found and fixed.

GUI interface performance and visual integration:

After data operation, the interface fails to be refreshed in time or freezes when visualized. The solution is to force the refresh method (update\_artifact\_view(), update\_visitor\_view()) to be called immediately after the data changes to ensure that the interface is updated in real time. For graph rendering, the optimization strategy is adopted: nx\_agraph.graphviz\_layout is prioritized (fallback on failure), FigureCanvasTkAgg is used to manage the canvas, and ax.clear() and fig.tight\_layout() are used to optimize the drawing performance and layout, which improves the user experience