Mystery Library Puzzle

Project Overview

The main goal is to design a game where players travel from a starting bookshelf to a final, mysterious book by solving puzzles along the way. The library's layout is not static; it is randomly created each time the game starts, offering a unique path for every playthrough. The game world is constructed as a network of connected "nodes," where each node represents a book on a shelf

Core Game Structure

The library is divided into distinct sections that guide the player's journey:

- **Entrance Shelves**: The game begins with the player choosing one of four starting shelves. Each of these shelves contains one "entrance" book.
- **Intermediate Shelves**: This is the main body of the library. Books in this section can be either "EASY" or "HARD," which determines the path forward.
- **Final Book**: The ultimate goal of the game is to reach this single, special book and solve its final puzzle.

Technical Requirements and Gameplay

To build this game, you must use a linked list as the fundamental data structure for the entire library. The connections between books are managed by pointers.

- **Book Node**: Each book is a Node in the linked list. It needs to contain its ID, type (Entrance, Intermediate, Final), puzzle clues, and pointers to the next node(s).
- **Branching Paths**: The game's navigation depends on the book's difficulty.
    - An **EASY** book has two pointers (next1 and next2), creating a fork in the road where the player must choose which path to take.
    - A **HARD** book has only one pointer (next1), forcing the player down a single, non-branching path.

- **Gameplay Flow**: The player starts at an entrance book, solves its puzzle, and then moves to the next book indicated by the pointers. This process continues until the final book is reached and its puzzle is solved to win the game.
- **Implementation Needs**: You will need to write functions for creating the randomized library, managing player movement through the nodes, and handling puzzle interactions. Proper memory management is crucial because you will be dynamically creating the book nodes.

1. The Concept: What is this Game?

Imagine a text-based adventure game where the "world" is a library.

- **The Goal:** The player starts at the entrance and must find their way to the **Final Mystery Book** to win.
- **The Obstacle:** To move from one book to the next, the player must solve a riddle or puzzle contained in their current book.
- **The Twist:** The library is not a normal building with hallways. It is a magical web of books. The path changes every time you play (it is randomized).

2. The Structure: The "Linked List" Library

In computer science terms, the library is a **Linked List**, and every book is a **Node**.

- **The Nodes (Books):**
  Instead of physical shelves, you have "Books." Each Book contains information (ID, puzzles) and points to the next Book(s).
- **The Connections (Pointers):**
  Books are connected by invisible strings (pointers). Book A points to Book B. You can only travel where the string points.

3. The Blueprint: The "Book" Node

You need to create a blueprint (a struct or class in C++) that defines what a "Book" is. Every single book in the game will have these properties:

- **ID:** A unique number (e.g., Book #101).
- **Type:** Is it an **ENTRANCE** (start), **INTERMEDIATE** (middle), or **FINAL** (end) book?
- **Difficulty:**

    - **EASY:** Has **2** clues and **2** paths forward (a fork in the road).
    - **HARD:** Has **1** clue and **1** path forward (a straight line).
- **Pointers (The "Next" Steps):**
    - next1: Points to the next book. (Used by all books).
    - next2: Points to a second optional book. (Used **only** by EASY books for the second path).
- **The Puzzle:** A storage container (like an array or vector) that holds the Question (Riddle) and the Answer (Solution).

4. The Library Layout (How it connects)

The game isn't one long line; it branches and flows.

1. **The Entrance (Start):**
    - You must create **4 specific books** that serve as starting points.
    - The player chooses one of these 4 to begin.
2. **The Middle (The Maze):**
    - This is a chain of Intermediate books.
    - If the current book is **EASY**, the player solves a puzzle and then chooses: "Do you want to go to path A (next1) or path B (next2)?"
    - If the current book is **HARD**, the player solves a puzzle and is automatically sent to the only path available (next1).
3. **The End (Target):**
    - All paths eventually lead to one single **Final Book**.
    - Once the player solves this book, the game ends.

5. The Mechanics: How the Game Plays

This is the logic you need to code for the "Game Loop":

1. **Setup:** When the program starts, it builds the library. It creates the nodes, randomly decides which are Easy/Hard, and randomly connects them together using pointers.
2. **Player Choice:** The player selects Entrance 1, 2, 3, or 4.

3. **Puzzle Loop:**
    - **Display:** Show the Book ID and the Clue/Riddle.
    - **Input:** Wait for the player to type an answer.
    - **Check:** Compare their answer to the correct solution stored in the Node.
    - **Fail:** If wrong, they try again (or lose, depending on your rules).
    - **Success:** If right, they move.
        - *Move Logic:* If it's an Easy book, ask them "Left or Right?" and follow that pointer. If it's Hard, follow the only pointer.
4. **Win Condition:** If the pointer leads to the **Final Book**, play the final puzzle. If they solve it, print "You Win!" and exit.

6. The Technical Requirements (C++ Checklist)

To make this happen, you need these specific programming tools:

- **struct or class**: To define the BookNode.
- **Pointers (BookNode* next1)**: Crucial. This is how you link book A to book B without them being next to each other in memory.
- **Dynamic Memory (new BookNode)**: You cannot list all variables at the start. You must use new to create books while the game is setting up.
- **Randomization (rand())**: You need to generate random numbers to decide if a book is Easy or Hard, and to scramble the connections so the game is different every time.
- **Input/Output (cin, cout)**: To talk to the player.
- **Input Validation**: You need code to ensure that if you ask for "1 or 2", the game doesn't crash if the player types "banana".

Summary of what you are building:

You are writing a C++ program that generates a random "maze" of linked objects (Books). The player "walks" through this maze by typing answers to riddles. The maze is held together by pointers (next1, next2), and the goal is to traverse from the head of the list (Entrance) to the tail (Final Book).

Members:

1. Game System Architect

- **Focus:** *Core Data Structures, Memory, Code Base Organization*
- **Responsibilities:**
    - Design the BookNode struct/class with fields: pointers (next1, next2), clues, ID, type, etc.
    - Establish linked list and branching structure for the library (entrance, intermediate, final nodes).
    - Ensure robust dynamic memory management (new, delete or smart pointers), properly clean up nodes.
    - Organize code base: create clear headers (.h) and sources (.cpp) for modular development.
    - Document the architecture and function interfaces for the team.

2. Gameplay & Logic Programmer

- **Focus:** *Game Rules, Puzzle Solving, Player Progression*
- **Responsibilities:**
    - Implement the main game loop: traversal from shelf selection to final book, including pathing logic.
    - Code player decisions: choosing shelves, picking branches at EASY nodes, and automatic movement at HARD nodes.
    - Develop puzzle/question presentation and checking (user answers, attempts).
    - Handle state tracking: which puzzles solved, current node, optional inventory/history tracking.
    - Integrate optional systems: hints, attempt limits, time-limited puzzles.
    - Validate player input throughout gameplay.

3. Randomization & World Builder

- **Focus:** *Library Generation, Difficulty Assignment, Clue Distribution*
- **Responsibilities:**
    - Code all randomization: assign book difficulties (EASY/HARD), shuffle order, create path connections so each run is unique but valid.
    - Populate nodes with clue/problem/solution data (from team content or designer).
    - Test and debug generation to ensure paths connect correctly from entrances to the final book.
    - Optionally, set up debugging/visualization tools to print or verify the random structure.
    - Configure parameters for replayability and balance.

4. User Interface & QA Engineer

- **Focus:** *Console I/O, Validation, Testing, Player Feedback*
- **Responsibilities:**
    - Build all console prompts: instructions for player choices, clue display, error handling.
    - Implement robust input validation: catch invalid responses, re-prompt, flush buffer, etc..
    - Refine UI/UX: make output clean, navigable, and user friendly.
    - Perform thorough functional and logic testing (manual, possibly automated): ensure correct gameplay, track/fix runtime errors and leaks.
    - Maintain game documentation, provide test feedback to team, help with bug fixes.

Files Structure:

```
MysteryLibraryPuzzle/
│
├── src/
│   ├── main.cpp
│   │
│   ├── core/                    [MEMBER 1: Game System Architect]
│   │   ├── BookNode.cpp
│   │   ├── BookNode.h
│   │   ├── Clue.cpp
│   │   ├── Clue.h
│   │   ├── MemoryManager.cpp
│   │   └── MemoryManager.h
│   │
│   ├── library/                 [MEMBER 1: Game System Architect]
│   │   ├── Library.cpp
│   │   ├── Library.h
│   │   ├── LibraryStructure.cpp
│   │   └── LibraryStructure.h
│   │
│   ├── game/                    [MEMBER 2: Gameplay & Logic Programmer]
│   │   ├── Game.cpp
│   │   ├── Game.h
│   │   ├── GameLoop.cpp
│   │   ├── GameLoop.h
```

```
│   │   ├── GameState.cpp
│   │   └── GameState.h
│   │
│   ├── player/                 [MEMBER 2: Gameplay & Logic Programmer]
│   │   ├── Player.cpp
│   │   ├── Player.h
│   │   ├── PlayerHistory.cpp
│   │   ├── PlayerHistory.h
│   │   ├── HistoryNode.cpp
│   │   ├── HistoryNode.h
│   │   ├── Inventory.cpp
│   │   ├── Inventory.h
│   │   ├── InventoryNode.cpp
│   │   └── InventoryNode.h
│   │
│   ├── puzzle/                 [MEMBER 2: Gameplay & Logic Programmer]
│   │   ├── PuzzleManager.cpp
│   │   ├── PuzzleManager.h
│   │   ├── PuzzleSolver.cpp
│   │   ├── PuzzleSolver.h
│   │   ├── AttemptTracker.cpp
│   │   ├── AttemptTracker.h
│   │   ├── HintSystem.cpp
│   │   ├── HintSystem.h
│   │   ├── HintNode.cpp
```

```
│   │   ├── HintNode.h
│   │   ├── Timer.cpp
│   │   └── Timer.h
│   │
│   ├── randomization/          [MEMBER 3: Randomization & World Builder]
│   │   ├── LibraryGenerator.cpp
│   │   ├── LibraryGenerator.h
│   │   ├── Randomizer.cpp
│   │   ├── Randomizer.h
│   │   ├── DifficultyAssigner.cpp
│   │   ├── DifficultyAssigner.h
│   │   ├── PathConnector.cpp
│   │   ├── PathConnector.h
│   │   ├── ClueDistributor.cpp
│   │   └── ClueDistributor.h
│   │
│   ├── ui/                     [MEMBER 4: UI & QA Engineer]
│   │   ├── Display.cpp
│   │   ├── Display.h
│   │   ├── Menu.cpp
│   │   ├── Menu.h
│   │   ├── InputHandler.cpp
│   │   ├── InputHandler.h
│   │   ├── InputValidator.cpp
│   │   ├── InputValidator.h
```

```
│   │   ├── MessageDisplay.cpp
│   │   └── MessageDisplay.h
│   │
│   └── utils/                  [SHARED BY ALL MEMBERS]
│       ├── FileHandler.cpp
│       ├── FileHandler.h
│       ├── Logger.cpp
│       ├── Logger.h
│       ├── DebugUtils.cpp
│       └── DebugUtils.h
│
├── include/                    [MEMBER 1: Organizes headers]
│   ├── core/
│   │   ├── BookNode.h
│   │   ├── Clue.h
│   │   └── MemoryManager.h
│   │
│   ├── library/
│   │   ├── Library.h
│   │   └── LibraryStructure.h
│   │
│   ├── game/
│   │   ├── Game.h
│   │   ├── GameLoop.h
│   │   └── GameState.h
```

```
│   │
│   ├── player/
│   │   ├── Player.h
│   │   ├── PlayerHistory.h
│   │   ├── HistoryNode.h
│   │   ├── Inventory.h
│   │   └── InventoryNode.h
│   │
│   ├── puzzle/
│   │   ├── PuzzleManager.h
│   │   ├── PuzzleSolver.h
│   │   ├── AttemptTracker.h
│   │   ├── HintSystem.h
│   │   ├── HintNode.h
│   │   └── Timer.h
│   │
│   ├── randomization/
│   │   ├── LibraryGenerator.h
│   │   ├── Randomizer.h
│   │   ├── DifficultyAssigner.h
│   │   ├── PathConnector.h
│   │   └── ClueDistributor.h
│   │
│   ├── ui/
│   │   ├── Display.h
```

```
│   │   ├── Menu.h
│   │   ├── InputHandler.h
│   │   ├── InputValidator.h
│   │   └── MessageDisplay.h
│   │
│   └── utils/
│       ├── FileHandler.h
│       ├── Logger.h
│       └── DebugUtils.h
│
├── data/                    [MEMBER 3: Populates content]
│   ├── puzzles/
│   │   ├── entrance_puzzles.txt
│   │   ├── easy_puzzles.txt
│   │   ├── hard_puzzles.txt
│   │   └── final_puzzles.txt
│   │
│   └── config/
│       ├── game_settings.txt
│       ├── difficulty_config.txt
│       └── randomization_params.txt
│
├── docs/                    [ALL MEMBERS contribute]
│   ├── README.md
│   ├── TEAM_STRUCTURE.md          [Overview of team responsibilities]
```

```
│   ├── ARCHITECTURE.md            [MEMBER 1: Architecture documentation]
│   ├── GAME_LOGIC.md              [MEMBER 2: Gameplay documentation]
│   ├── RANDOMIZATION.md           [MEMBER 3: Generation documentation]
│   ├── UI_GUIDE.md                [MEMBER 4: UI/UX documentation]
│   ├── API_REFERENCE.md           [MEMBER 1: Function interfaces]
│   ├── TESTING_GUIDE.md           [MEMBER 4: Testing procedures]
│   └── CLASS_DIAGRAM.png          [MEMBER 1: Visual architecture]
│
├── tests/                         [MEMBER 4: Primary, ALL assist]
│   ├── unit/
│   │   ├── test_booknode.cpp      [MEMBER 1]
│   │   ├── test_clue.cpp          [MEMBER 1]
│   │   ├── test_memory.cpp        [MEMBER 1]
│   │   ├── test_library.cpp       [MEMBER 1]
│   │   ├── test_game_loop.cpp     [MEMBER 2]
│   │   ├── test_player.cpp        [MEMBER 2]
│   │   ├── test_puzzle_solver.cpp [MEMBER 2]
│   │   ├── test_inventory.cpp     [MEMBER 2]
│   │   ├── test_history.cpp       [MEMBER 2]
│   │   ├── test_hint_system.cpp   [MEMBER 2]
│   │   ├── test_timer.cpp         [MEMBER 2]
│   │   ├── test_randomizer.cpp    [MEMBER 3]
│   │   ├── test_library_generator.cpp [MEMBER 3]
│   │   ├── test_difficulty_assigner.cpp [MEMBER 3]
│   │   ├── test_path_connector.cpp [MEMBER 3]
```

```
│   │   ├── test_input_validator.cpp   [MEMBER 4]
│   │   ├── test_display.cpp         [MEMBER 4]
│   │   └── test_menu.cpp            [MEMBER 4]
│   │
│   ├── integration/
│   │   ├── test_full_game.cpp       [MEMBER 4 leads, ALL assist]
│   │   ├── test_library_to_game.cpp   [MEMBER 2 & 3]
│   │   ├── test_puzzle_flow.cpp      [MEMBER 2 & 4]
│   │   └── test_generation_valid.cpp  [MEMBER 3 & 1]
│   │
│   ├── debugging/                [MEMBER 3: Visualization tools]
│   │   ├── visualize_library.cpp
│   │   ├── print_structure.cpp
│   │   └── path_validator.cpp
│   │
│   └── test_runner.cpp             [MEMBER 4: Test orchestration]
│
├── build/
│   └── (compiled object files)
│
├── bin/
│   └── (executable)
│
├── logs/                       [MEMBER 4: QA logs]
│   ├── test_results/
```

```
│   ├── bug_reports/
│   └── performance_logs/
│
├── team/                       [Team coordination]
│   ├── meetings/
│   │   └── meeting_notes.md
│   ├── task_assignments.md
│   ├── progress_tracker.md
│   └── integration_schedule.md
│
├── Makefile                    [MEMBER 1: Build system]
├── .gitignore
└── README.md
```