ĐẠI HỌC ĐÀ NẴNG

ĐẠI HỌC BÁCH KHOA

KHOA CÔNG NGHỆ THÔNG TIN

PBL2. DỰ ÁN CƠ SỞ LẬP TRÌNH

ĐỀ TÀI: XÂY DỰNG VÀ PHÁT TRIỂN GAME CHIẾN THUẬT TRÍ TUỆ "KỲ MÔN THẦN TỐC"

GIẢNG VIÊN HƯỚNG DẪN: TS. PHẠM MINH TUẤN

SINH VIÊN THỰC HIỆN:

MAI TẠ TRÚC VY (102240068) LỚP: 24T_DT1

HOÀNG TRẦN ĐỨC HẢI (102240305) LỚP: 24T_Nhat1

Đà Nẵng, tháng 12 / 2025

LỜI MỞ ĐẦU

Trong kỷ nguyên chuyển đổi số, Công nghệ Thông tin không chỉ là một lĩnh vực khoa học thuần túy, mà còn là một công cụ mạnh mẽ để chuyển hóa những ý tưởng thành hiện thực. Trò chơi điện tử (game), với vai trò vừa là một sản phẩm giải trí, vừa là một ứng dụng kỹ thuật phức tạp, thể hiện rõ nhất sự kết hợp giữa lý thuyết thuật toán, cấu trúc dữ liệu và khoa học thiết kế phần mềm. Chính nơi giao điểm này, những ý tưởng sáng tạo được biến thành trải nghiệm tương tác có giá trị thực tiễn.

Đồ án Cơ sở Lập trình (PBL2) đại diện cho một cột mốc quan trọng trong hành trình học tập của sinh viên Công nghệ Thông tin. Học phần này đòi hỏi sinh viên phải tổng hợp và ứng dụng sâu sắc kiến thức từ hai lĩnh vực nền tảng: **Lập trình Hướng đối tượng (OOP)** và **Cấu trúc dữ liệu & Giải thuật**, vào việc xây dựng một sản phẩm phần mềm hoàn chỉnh và có giá trị thực tiễn. Nhận thức được tầm quan trọng đó, nhóm chúng em đã lựa chọn thực hiện đề tài **"Xây dựng và Phát triển Game Chiến Thuật Trí Tuệ – Kỳ Môn Thần Tốc"** để chứng minh sự am hiểu sâu sắc về cả chiều sâu lý thuyết cũng như khả năng ứng dụng thực tiễn. Thay vì xây dựng một ứng dụng quản lý thông thường, dự án này tập trung vào việc triển khai một bộ não nhân tạo (AI) thông minh, kết hợp thuật toán **Tìm kiếm theo Chiều rộng (BFS)** và **Quy hoạch động với Bitmask** để giải quyết bài toán tối ưu hóa, từ đó tạo ra một trò chơi có thách thức thực sự.

Trong quá trình nghiên cứu, thiết kế và triển khai dự án, nhóm chúng em đã nỗ lực sâu sắc để nắm vững các nguyên tắc lập trình, từ các mẫu thiết kế cổ điển (Singleton, State) cho đến những kỹ thuật tối ưu hóa hiệu năng. Mỗi thành phần của hệ thống – từ việc quản lý dữ liệu bản đồ, xử lý input từ người chơi, cho đến việc triển khai công cụ AI – đều được suy tính kỹ lưỡng để đảm bảo tính chính xác, hiệu quả và khả năng mở rộng.

Tại đây, nhóm chúng em xin gửi lời cảm ơn sâu sắc đến **Khoa Công nghệ Thông tin, Trường Đại học Bách Khoa – Đại học Đà Nẵng** đã tạo dựng một môi trường học tập tích cực, với cơ sở vật chất và giáo viên tuyệt vời. Đặc biệt, chúng em xin bày tỏ lòng biết ơn chân thành đến **TS. Phạm Minh Tuấn** – giảng viên hướng dẫn của dự án – với sự tận tình chỉ bảo, những gợi ý quý báu, và các hướng dẫn chi tiết từ giai đoạn khởi động cho đến hoàn thành. Thầy đã giúp nhóm chúng em không chỉ hiểu sâu các khái niệm mà còn biết cách vận dụng chúng một cách khôn ngoan vào thực tiễn.

Mặc dù chúng em đã nỗ lực tối đa, nhưng do hạn chế về kiến thức, kinh nghiệm và điều kiện, sản phẩm này không tránh khỏi những thiếu sót. Chúng em hân hạnh tiếp nhận những nhận xét, đề xuất và phê bình xây dựng từ thầy cô và các bạn để hoàn thiện dự án trong tương lai. Báo cáo này được biên soạn với nỗ lực cao nhất nhằm trình bày một cách rõ ràng, khoa học và hệ thống toàn bộ quá trình thực hiện, từ phân tích bài toán, thiết kế cấu trúc, triển khai thuật toán, cho đến đánh giá kết quả cuối cùng.

Chúng em xin chân thành cảm ơn!

MỤC LỤC

DANH MỤC HÌNH VẼ

PHẦN 1: TỔNG QUAN ĐỀ TÀI

1.1 Tổng quan đề tài

1.1.1 Tên đề tài

Xây dựng và Phát triển Game Chiến Thuật Trí Tuệ – Kỳ Môn Thần Tốc

1.1.2 Mục tiêu thực hiện đề tài

- **Xây dựng nền tảng tư duy chiến thuật:** Tạo ra một không gian nơi người chơi phải vận dụng khả năng quan sát tổng thể và phân tích logic. Mỗi màn chơi là một bài toán tối ưu, buộc người chơi phải lập kế hoạch cho từng bước đi để đạt được hiệu quả cao nhất (Rank S).
- **Ứng dụng kỹ thuật lập trình nâng cao:** Vận dụng ngôn ngữ **C++** và tư duy lập trình hướng đối tượng (**OOP**) để xây dựng cấu trúc mã nguồn sạch, dễ mở rộng thông qua các Design Patterns như Singleton hay Parameter Object.
- Làm chủ thư viện **SDL2** để quản lý vòng đời ứng dụng, từ xử lý sự kiện, âm thanh đến kết xuất đồ họa đa tầng.
- **Thiết lập chuẩn mực tối ưu bằng AI:** Thay vì để người chơi tự tìm đường một cách cảm tính, đề tài hướng tới việc sử dụng trí tuệ nhân tạo để tính toán "con đường tuyệt đối" (Thiên Mệnh). Điều này không chỉ tạo ra sự thách thức mà còn giúp người chơi có một hệ quy chiếu chính xác để tự đánh giá tư duy của bản thân.
- **Hoàn thiện trải nghiệm người dùng (UX):** Tích hợp các tính năng bổ trợ như hệ thống **Undo (Hoàn tác)** dựa trên cấu trúc dữ liệu Stack và cơ chế **Auto-scaling** bản đồ, giúp người chơi tập trung hoàn toàn vào việc giải đố mà không bị rào cản về kỹ thuật hay giao diện.

1.1.3 Giới thiệu về các thuật toán

- ***Thuật toán tìm đường theo chiều rộng (Breadth-First Search - BFS)***

Trong một bản đồ dạng lưới (Grid-map) với nhiều vật cản, việc xác định khoảng cách ngắn nhất giữa hai điểm không đơn thuần là đường chim bay. BFS được sử dụng như một "máy quét" không gian, đảm bảo tìm ra con đường ngắn nhất thực tế giữa vị trí xuất phát và các Trận Nhãn (Shrines). Đây là bước đệm quan trọng để xây dựng Ma trận khoảng cách (Distance Matrix) cho hệ thống.

- **Quy hoạch động với Bitmask giải bài toán người du lịch (TSP with Bitmask DP)**

Linh hồn của "Kỳ Môn Thần Tốc" nằm ở việc tìm ra thứ tự đi qua tất cả các điểm mục tiêu sao cho tổng số bước là nhỏ nhất. Thay vì sử dụng phương pháp vét cạn (Brute-force) có độ phức tạp O(N!), thuật toán Quy hoạch động kết hợp với kỹ thuật Bitmask (sử dụng các bit của số nguyên để biểu diễn tập hợp các điểm đã đi qua) giúp tối ưu hóa thời gian tính toán xuống còn O(2^N x N^2).

⇨ Sự kết hợp này cho phép AI đưa ra lời giải cho các mê cung phức tạp chỉ trong tích tắc, tạo nên một chuẩn mực "Thần Tốc" mà người chơi phải nỗ lực để chạm tới.

1.1.4 Cấu trúc chương trình

Chương trình được xây dựng theo kiến trúc phân tầng, tách biệt rõ ràng giữa dữ liệu, logic xử lý và hiển thị đồ họa. Việc áp dụng mẫu thiết kế **Singleton** giúp quản lý tập trung các tài nguyên hệ thống, tránh lãng phí bộ nhớ và xung đột dữ liệu. Cấu trúc gồm 4 phân hệ chính:

- **Tầng Cốt lõi (Core Logic):** Bao gồm GameEngine điều phối vòng đời game và Config lưu trữ các hằng số hệ thống. Đây là nơi duy trì nhịp đập của trò chơi thông qua vòng lặp Game Loop.
- **Tầng Thực thể (Entities):** Quản lý đối tượng nhân vật (Player) và hệ thống bản đồ (Map). Các đối tượng này kế thừa từ lớp trừu tượng GameObject, đảm bảo tính đa hình trong lập trình hướng đối tượng.
- **Tầng Tài nguyên (Resource Management):** Gồm TextureManager và SoundManager, chịu trách nhiệm nạp, lưu trữ và giải phóng hình ảnh, âm thanh, phông chữ một cách tự động.
- **Tầng Thuật toán (Algorithms):** Đóng vai trò là "bộ não" với ThienCoEngine, nơi thực thi các phép toán BFS và DP Bitmask để tìm ra lời giải tối ưu.

1.1.5 Tính năng và giao diện

Trò chơi được thiết kế nhằm tối ưu hóa sự tương tác giữa tư duy người chơi và phản hồi của hệ thống:

- **Tính năng Gameplay:**
  - **Hệ thống Undo (Hoàn tác):** Sử dụng cấu trúc dữ liệu Stack để lưu trữ các "Moment" (trạng thái tạm thời), cho phép người chơi rút kinh nghiệm từ những bước đi sai lầm.
  - **Tính toán Rank S/A:** Hệ thống tự động so sánh số bước đi của người chơi với kết quả của AI để xếp hạng, tạo động lực chinh phục sự hoàn hảo.

- **Auto-scaling:** Bản đồ tự động co giãn và căn giữa tùy theo kích thước lưới, đảm bảo trải nghiệm thị giác luôn cân đối.

- **Giao diện người dùng (UI/UX):**
  - **Phong cách cổ điển:** Sử dụng đồ họa Tilemap kết hợp với hiệu ứng chuyển cảnh Fade In/Out mượt mà.
  - **Hệ thống cốt truyện:** Lồng ghép các màn hình hội thoại giữa nhân vật và "Thầy" (Sensei) trước mỗi màn chơi để tăng tính dẫn dắt và triết lý.
  - **Phản hồi trực quan:** Hiệu ứng nhân vật mờ dần và rơi xuống vực khi đi vào ô "Air" giúp người chơi nhận diện lỗi sai một cách sinh động.

1.1.6 Công cụ lập trình

Để xây dựng một sản phẩm có tính ổn định và hiệu năng cao, các công cụ sau đã được lựa chọn:

- **Ngôn ngữ lập trình: C++** (Tiêu chuẩn hiện đại), lựa chọn tối ưu để rèn luyện tư duy quản lý con trỏ và cấu trúc dữ liệu phức tạp.
- **Thư viện đồ họa: SDL2** (Simple Direct Media Layer) cùng các thư viện mở rộng như SDL_image, SDL_mixer, và SDL_ttf để xử lý hình ảnh PNG, âm thanh MP3 và phông chữ TrueType.
- **Môi trường phát triển (IDE):** Visual Studio, kết hợp với bộ công cụ biên dịch MinGW/GCC.
- **Quản lý phiên bản:** Git/GitHub để theo dõi lịch sử thay đổi mã nguồn và đảm bảo an toàn dữ liệu trong quá trình phát triển.

1.1.7 Ý nghĩa và ứng dụng

Đồ án "Kỳ Môn Thần Tốc" không dừng lại ở một sản phẩm giải trí đơn thuần mà mang trong mình những giá trị thực tiễn:

- **Về giáo dục:** Là công cụ rèn luyện tư duy thuật toán và khả năng tối ưu hóa cho người chơi. Trò chơi buộc não bộ phải hình thành thói quen "tính toán trước nhiều bước" (Look-ahead), một kỹ năng quan trọng trong cả lập trình lẫn đời sống.
- **Về học thuật:** Minh chứng cho việc ứng dụng thành công các thuật toán phức tạp như Quy hoạch động và Bitmask vào sản phẩm thực tế. Đồ án giúp làm sáng tỏ khoảng cách giữa lý thuyết khô khan và ứng dụng sinh động.
- **Về ứng dụng:** Cấu trúc Engine được xây dựng trong đồ án có khả năng mở rộng cao, có thể phát triển thành các ứng dụng mô phỏng tìm đường, quản lý kho bãi hoặc các trò chơi giải đố phức tạp hơn trong tương lai.

1.2 Cơ sở lý thuyết

1.2.1 Ý tưởng thực hiện

Ý tưởng của "Kỳ Môn Thần Tốc" được khơi nguồn từ sự giao thoa giữa **trò chơi giải đố cổ điển** và **toán học tối ưu**. Trong cuộc sống, chúng ta luôn đối mặt với những lựa chọn: "Đi đường nào là nhanh nhất?", "Làm việc gì trước là hiệu quả nhất?". Chúng em muốn biến những câu hỏi đó thành một trải nghiệm tương tác sinh động.

- **Sự thách thức từ mê cung:** Mê cung không chỉ là những bức tường, nó là đại diện cho những rào cản trong tư duy. Ý tưởng cốt lõi là tạo ra một môi trường lưới (Grid) nơi mỗi ô gạch là một trạng thái, buộc người chơi phải cân nhắc giữa việc "đi để thu thập" và "đi để tồn tại".
- **Sự chuẩn xác của máy tính làm hệ quy chiếu:** Thay vì chỉ tạo ra một trò chơi vượt ải thông thường, ý tưởng đột phá là tích hợp một "Giải pháp hoàn hảo" chạy ngầm. Người chơi sẽ không chỉ chơi với máy, mà là đang theo đuổi sự tối ưu mà thuật toán đã vạch ra. Đó là sự khao khát đạt đến cái gọi là "Thiên Mệnh" – con đường không thể ngắn hơn được nữa.
- **Triết lý về sự hoàn tác (Undo):** Trong mê cung của cuộc đời, sai lầm đôi khi phải trả giá đắt. Nhưng trong "Kỳ Môn Thần Tốc", chúng em cho phép người chơi quay lại. Ý tưởng này dựa trên phương pháp "Thử và Sai" (Trial and Error) – khuyến khích người chơi dám thử những chiến thuật táo bạo nhất mà không sợ bị bế tắc.

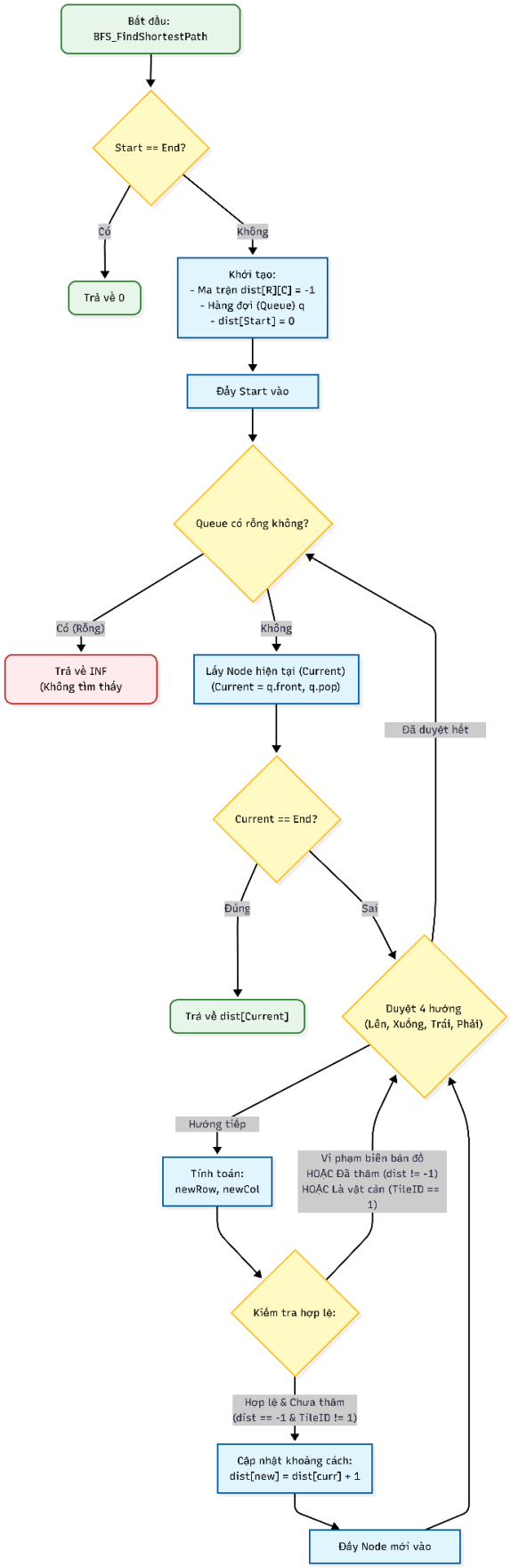
1.2.2 Cơ sở lý thuyết

Để xây dựng nên một hệ thống vận hành ổn định và thông minh, đồ án dựa trên các nền tảng lý thuyết khoa học máy tính vững chắc:

a) Lý thuyết đồ thị và Thuật toán tìm đường (Graph Theory)

Bản đồ game được mô hình hóa dưới dạng một đồ thị vô hướng, trong đó mỗi ô gạch là một đỉnh (Vertex) và các ô kề nhau được nối bởi một cạnh (Edge).

- **BFS (Breadth-First Search):** Dựa trên cơ sở lý thuyết về việc duyệt đồ thị theo từng mức (Level-order traversal), BFS đảm bảo tìm ra đường đi ngắn nhất về mặt số cạnh trong một đồ thị không trọng số. Đây là nền tảng để xác định chi phí di chuyển thực tế trên địa hình bản đồ.

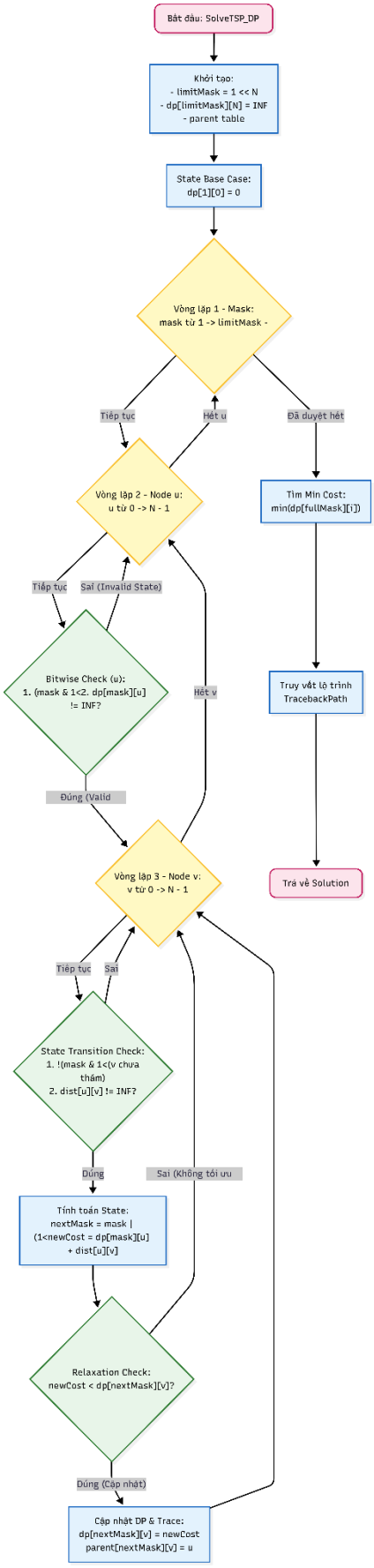


Hình 1: Sơ đồ khối thuật toán BFS

b) Bài toán Người du lịch (TSP)

Đây là một trong những bài toán nổi tiếng nhất trong lý thuyết tối ưu hóa tổ hợp. Mục tiêu là tìm lộ trình ngắn nhất đi qua một tập hợp các điểm cho trước và kết thúc tại một điểm xác định.

- **Quy hoạch động (Dynamic Programming):** Thay vì tính toán lại nhiều lần các bài toán con, chúng em sử dụng lý thuyết lưu trữ kết quả của các trạng thái đã đi qua.
- **Trạng thái Bitmask:** Một kỹ thuật học thuật sử dụng số nguyên ở hệ nhị phân để biểu diễn các tập hợp. Ví dụ: một số nguyên có giá trị 11 (1011 nhị phân) đại diện cho việc đã đi qua các điểm thứ 0, 1 và 3. Kỹ thuật này giúp chuyển đổi các bài toán có độ phức tạp giai thừa O(N!) về độ phức tạp mũ O(2^N . N^2), cho phép xử lý mượt mà trong môi trường thời gian thực của game.



Hình 2: Sơ đồ khối thuật toán quy hoạch động (DP Bitmask)

c) Mô hình lập trình hướng đối tượng (OOP)

Toàn bộ mã nguồn được xây dựng trên 4 trụ cột của OOP:

- **Tính đóng gói:** Giấu đi các chi tiết xử lý của Engine và chỉ đưa ra các interface cần thiết.
- **Tính kế thừa:** Sử dụng lớp cơ sở GameObject để tái sử dụng mã cho Player, Shrines.
- **Tính đa hình:** Cho phép các đối tượng khác nhau có cách Update và Render riêng biệt.
- **Tính trừu tượng:** Mô hình hóa các khái niệm trừu tượng như "Trạng thái game" hay "Vòng đời đối tượng" thành các class quản lý.

d) Nguyên lý Render và Xử lý âm thanh trong SDL2

Dựa trên lý thuyết về Double Buffering (Bộ đệm kép) để tránh hiện tượng nhấp nháy màn hình và Frame Rate Limiting để giữ cho trò chơi chạy ổn định ở tốc độ 60 khung hình/giây, đảm bảo sự mượt mà trong mọi thao tác của người dùng.

PHẦN 2: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

2.1 Phân tích chức năng hệ thống

Hệ thống của "Kỳ Môn Thần Tốc" được phân rã thành các nhóm chức năng cốt lõi, đảm bảo sự phối hợp nhịp nhàng giữa tương tác người dùng và tính toán logic của máy tính.

2.1.1. Chức năng Quản lý trò chơi (Core Management)

Đây là tầng điều khiển trung tâm, nơi duy trì sự tồn tại và luồng vận hành của ứng dụng:

- **Quản lý vòng đời (Game Loop):** Đảm bảo chu trình *Xử lý sự kiện -> Cập nhật logic -> Kết xuất đồ họa* diễn ra liên tục ở tốc độ 60 FPS.
- **Quản lý trạng thái (State Management):** Điều phối việc chuyển đổi mượt mà giữa các màn hình: Intro, Story, Gameplay, và Result thông qua hệ thống chuyển cảnh (Fade Effect).
- **Quản lý tài nguyên (Resource Management):** Tự động hóa việc nạp và giải phóng bộ nhớ cho hình ảnh, âm thanh và phông chữ. Cơ chế "Lazy Loading" giúp tiết kiệm RAM bằng cách chỉ nạp những gì cần thiết cho màn chơi hiện tại.

2.1.2 Chức năng Gameplay và Tương tác

Nhóm chức năng này trực tiếp tạo nên trải nghiệm người dùng trên bản đồ:

- **Di chuyển theo lưới (Grid-based Movement):** Nhân vật không di chuyển tự do mà luôn khớp vào các ô gạch. Chức năng này bao gồm việc nội suy vị trí (Interpolation) để tạo cảm giác trượt mượt mà giữa các ô.
- **Xử lý va chạm và Địa hình:** Nhận diện thuộc tính của từng ô lưới. Nếu là "Trận Nhãn" (Shrine), hệ thống ghi nhận thu thập; nếu là "Chân Không" (Air), kích hoạt trạng thái rơi (Falling).
- **Hệ thống Hoàn tác (Undo System):** Chức năng đặc biệt cho phép "quay ngược thời gian". Hệ thống trích xuất trạng thái từ Stack để khôi phục vị trí nhân vật và trạng thái các vật phẩm trên bản đồ về đúng khoảnh khắc trước đó.

2.1.3 Chức năng Trí tuệ nhân tạo (ThienCoEngine)

Đây là chức năng "vàng" mang tính học thuật cao của đồ án:

- **Phân tích bản đồ:** Tự động quét toàn bộ địa hình để xác định các điểm quan trọng.
- **Tính toán Ma trận khoảng cách:** Sử dụng BFS để tìm đường ngắn nhất giữa từng cặp điểm, loại bỏ các vật cản để có dữ liệu chi phí di chuyển chính xác.
- **Giải mã "Thiên Mệnh":** Thực thi thuật toán DP Bitmask trên ma trận đã tính để tìm ra chu trình tối ưu qua tất cả các điểm. Kết quả này được lưu lại làm "mốc Rank S" để so sánh với kết quả của người chơi.

2.1.4 Chức năng Phản hồi trực quan và Âm thanh

Chức năng này đóng vai trò "giao tiếp" với cảm xúc người chơi:

- **Đồ họa đa lớp:** Kết xuất nền Parallax, bản đồ Tilemap và nhân vật theo đúng thứ tự lớp (Layering) để tạo chiều sâu không gian.
- **Cơ chế Auto-scaling:** Tự động tính toán tỷ lệ màn hình để bản đồ luôn nằm chính giữa, bất kể kích thước mê cung thay đổi theo từng màn.
- **Hệ thống âm thanh động:** Phân phối hiệu ứng âm thanh (SFX) cho từng hành động (bước đi, ăn điểm, rơi) và nhạc nền (BGM) thay đổi theo trạng thái thắng/thua.

2.2 Sơ đồ phân cấp chức năng (Functional Decomposition Diagram)

Để hình dung rõ hơn, các chức năng trên được tổ chức theo sơ đồ hình cây, từ các module lớn đến các nhiệm vụ chi tiết:

2.2.1. Module Hệ thống (System Management)

- Khởi tạo & Giải phóng: Khởi tạo SDL2, Window, Renderer; Giải phóng bộ nhớ khi thoát.
- Quản lý Tài nguyên: Nạp và lưu trữ Texture (hình ảnh), SFX/BGM (âm thanh), Font (chữ).
- Quản lý Vòng lặp (Game Loop): Kiểm soát Delta Time, giới hạn tốc độ khung hình (FPS Limiter).
- Xử lý sự kiện (Input Handling): Tiếp nhận tín hiệu từ bàn phím (di chuyển, Undo, thoát).

2.2.2. Module Đồ họa & Hiển thị (Graphics & Rendering)

- Kết xuất thực thể: Vẽ bản đồ (Tilemap), vẽ nhân vật (Sprite Animation).
- Hiệu ứng hình ảnh: Hiệu ứng Fade (chuyển cảnh), hiệu ứng rơi tự do (Scaling/Alpha), căn giữa bản đồ tự động (Auto-scaling).

- Giao diện người dùng (UI): Hiển thị số bước chân, trạng thái Rank S/A, màn hình thông báo cốt truyện.

2.2.3. Module Logic Gameplay (Core Logic)

- Điều khiển thực thể: Di chuyển theo lưới (Grid-based Movement), nội suy vị trí mượt mà.
- Xử lý va chạm: Kiểm tra thuộc tính ô gạch (Đất, Vực, Trận nhãn).
- Quản lý trạng thái chơi: Tính toán số bước đi, theo dõi tiến trình thu thập, kiểm tra điều kiện Thắng/Thua.
- Hệ thống Hoàn tác (Undo): Lưu trữ trạng thái (Stack), khôi phục vị trí và dữ liệu bản đồ.
- Quản lý màn chơi: Tải dữ liệu từ file .txt, chuyển cấp độ (Level Switching).

2.2.4. Module Trí tuệ nhân tạo (AI Engine - ThienCo)

- Phân tích địa hình: Sử dụng BFS quét bản đồ để tính ma trận khoảng cách giữa các điểm quan trọng.
- Tối ưu hóa hành trình: Thực thi thuật toán DP Bitmask để giải bài toán TSP.
- Thiết lập chuẩn mực: Xuất ra số bước đi tối ưu nhất để làm mốc so sánh xếp hạng (Ranking System).

2.2.5. Module Âm thanh (Audio System)

- Nhạc nền (BGM): Phát nhạc lặp lại theo trạng thái màn hình (Menu, In-game, Result).
- Hiệu ứng âm thanh (SFX): Phát âm thanh tương ứng khi bước đi, thu thập vật phẩm, hoặc thất bại.

PHẦN 3: THIẾT KẾ CẤU TRÚC DỮ LIỆU

3.1 Phát biểu bài toán

Hệ thống trò chơi "Kỳ Môn Thần Tốc" là sự tổng hòa của một chuỗi các vấn đề tối ưu hóa. Để xây dựng được chương trình hoàn thiện, chúng ta cần giải quyết tuần tự 3 bài toán con, tương ứng với 3 giai đoạn giác ngộ của một Kỳ Môn Sư:

Quan sát (Minh Trí) ⟶ Tính toán (Định Mệnh) ⟶ Hành động (Nhập Thế)

- **Tổng quan bài toán thực tế (Chương trình hoàn thiện)**
- **Mục tiêu:** Điều khiển nhân vật di chuyển từ điểm xuất phát (1,1), đi qua (Khai mở) tất cả N Trận Nhãn trên bản đồ lưới R times C.
- **Điều kiện Thắng/Thua:**
  - **Thiên Cơ (AI):** Ngay khi bắt đầu, hệ thống chạy ngầm thuật toán để tìm ra số bước đi tối thiểu tuyệt đối S.
  - **Chiến thắng:** Người chơi hoàn thành nhiệm vụ với số bước Steps ≤ S trong thời gian quy định.
  - **Thất bại:** Số bước đi vượt quá S hoặc hết giờ.
- **Cơ chế điều khiển:**
  - **Di chuyển:** 4 hướng (Trên, Dưới, Trái, Phải).
  - **Hồi Tưởng (Undo):** Sử dụng ngăn xếp để quay lui trạng thái.

3.2 Phân tích và ứng dụng cấu trúc dữ liệu

Để giải quyết 3 bài toán trên, đồ án đã lựa chọn và thiết kế các cấu trúc dữ liệu đặc thù, đảm bảo tối ưu cả về không gian lưu trữ lẫn thời gian truy xuất.

3.2.1 Giải quyết Bài toán 1: Minh Trí - Xây dựng Ma trận Tri thức

Giới hạn thời gian: **1.0 giây**

Giới hạn bộ nhớ: **512 MB**

Giới thiệu

Trước khi đối mặt với Thiên Cơ, một Kỳ Môn Sư phải đạt đến cảnh giới "Minh Trí" - thấu hiểu bản chất của vạn vật. Bước đầu tiên không phải là hành động, mà là quan sát. Bằng cách thiền định và kết nối với dòng chảy năng lượng của trận đồ, Kỳ Môn Sư phải cảm nhận và ghi lại "linh mạch" - những con đường ngắn nhất kết nối các điểm huyệt trọng yếu. Đây là tri thức nền tảng, là tấm bản đồ của chính vận mệnh.

Đề bài

- Cho một Thiên Không Trận Đồ dưới dạng lưới R x C, bao gồm các ô đất trống ('.'), núi non hiểm trở ('#'), và N Trận Nhãn ('S'). Kỳ Môn Sư bắt đầu hành trình tại ô (1, 1). Từ một ô, họ có thể di chuyển đến các ô kề cạnh (trên, dưới, trái, phải), miễn là ô đó không phải là núi.
- Nhiệm vụ của bạn là tính toán và xây dựng một ma trận tri thức, chứa số bước đi ngắn nhất giữa mọi cặp địa điểm quan trọng. Các địa điểm quan trọng bao gồm điểm xuất phát và N Trận Nhãn.

Input

- Dòng đầu tiên chứa ba số nguyên dương R, C, và N ($1 \leq R, C \leq 100$, $1 \leq N \leq 15$).
- R dòng tiếp theo, mỗi dòng chứa C ký tự, mô tả trận đồ.
- Dữ liệu đảm bảo có đúng N ký tự 'S', ô (1, 1) luôn là một ô đất trống.

Output

In ra một ma trận kích thước (N + 1) x (N + 1).

- Các địa điểm được đánh số từ 0 đến N: **Địa điểm 0** là điểm xuất phát (1, 1), **địa điểm 1 đến N** là các Trận Nhãn theo thứ tự xuất hiện từ trên xuống dưới, từ trái qua phải.
- Giá trị tại hàng i, cột j là số bước đi ngắn nhất từ địa điểm i đến địa điểm j. Nếu không có đường đi, in ra -1.

Ví dụ

Input

```
3 6 3
.S..#.
..#.#.
S.#S..
```

Output

```
0 1 2 5
1 0 3 4
2 3 0 7
5 4 7 0
```

Giải thích ví dụ

- Địa điểm 0: (1, 1). Địa điểm 1: (1, 2). Địa điểm 2: (3, 1). Địa điểm 3: (3, 4).
- Ma trận đầu ra thể hiện tri thức về khoảng cách ngắn nhất giữa các cặp địa điểm.

Mục tiêu: Tính khoảng cách ngắn nhất giữa mọi cặp điểm quan trọng (Xuất phát + *N* Trận Nhãn) trên bản đồ có vật cản.

Mô hình hóa dữ liệu:

- Bản đồ được xem là một đồ thị vô hướng dạng lưới (Grid Graph).
- Các ô đất trống ('.') và Trận Nhãn ('S') là các đỉnh (Vertex).
- Hai ô kề nhau không phải là núi ('#') có một cạnh nối (Edge) với trọng số là 1.

Cấu trúc dữ liệu sử dụng:

- **Queue (std::queue<Point>):** Dùng cho thuật toán **BFS (Breadth-First Search)**. Vì đồ thị không có trọng số (mỗi bước đi đều tốn 1 đơn vị), Queue giúp duyệt các ô theo từng lớp lan rộng (như sóng nước), đảm bảo tìm ra đường đi ngắn nhất đầu tiên.
- **Mảng 2 chiều (int dist[R][C]):** Lưu trữ khoảng cách từ điểm nguồn đến tất cả các điểm khác trên bản đồ trong một lần chạy BFS.
- **Ma trận kề (vector<vector<int>> adjacencyMatrix):** Đây là kết quả đầu ra (Output). Ma trận kích thước (N+1) x (N+1) lưu trữ khoảng cách ngắn nhất giữa các điểm quan trọng với nhau.

Đánh giá: Với giới hạn R, C ≤ 100, độ phức tạp của mỗi lần BFS là O(R x C). Chúng ta chạy BFS N+1 lần, tổng độ phức tạp là O(N x R x C), hoàn toàn thỏa mãn giới hạn thời gian 1.0s.

3.2.2 Giải quyết Bài toán 2: Định Mệnh - Tìm kiếm Thiên Mệnh (TSP)

Giới hạn thời gian: **1.0 giây**

Giới hạn bộ nhớ: **512 MB**

Giới thiệu

Khi đã có trong tay ma trận tri thức, Kỳ Môn Sư bước vào giai đoạn "Định Mệnh". Đây là lúc trí tuệ chiến thuật được phát huy đến cực hạn. Bằng cách suy diễn và tính toán, Kỳ Môn Sư phải vạch ra con đường hoàn hảo nhất, một lộ trình duy nhất đi qua tất cả các Trận Nhãn với tổn hao ít năng lượng (bước

đi) nhất. Con đường này được gọi là "Thiên Mệnh" - lộ trình tối thượng mà chính ý thức của trận đồ đã định sẵn.

Đề bài

Bạn được cung cấp ma trận tri thức từ Bài 1, chứa khoảng cách giữa N + 1 địa điểm quan trọng (0 là điểm xuất phát, 1 đến N là các Trận Nhãn). Dựa vào ma trận này, hãy tìm một lộ trình bắt đầu từ địa điểm 0, đi qua tất cả N Trận Nhãn (mỗi nơi đúng một lần) sao cho tổng quãng đường di chuyển là ngắn nhất có thể.

Input

- Dòng đầu tiên chứa một số nguyên dương N ($1 \leq N \leq 15$).
- N + 1 dòng tiếp theo, mỗi dòng chứa N + 1 số nguyên, tạo thành ma trận khoảng cách dist ($-1 \leq dist[i][j] \leq 10000$).
- dist[i][j] là chi phí di chuyển từ địa điểm i đến j.

Output

Nếu không tồn tại lộ trình thỏa mãn, in ra -1. Ngược lại, in ra hai dòng:

- Dòng đầu tiên: một số nguyên là tổng số bước đi tối thiểu của lộ trình Thiên Mệnh.
- Dòng thứ hai: N + 1 số nguyên, là thứ tự các địa điểm được khai mở trong lộ trình tối ưu, bắt đầu bằng 0.

Ví dụ

Input

```
3
0 1 2 5
1 0 3 4
2 3 0 7
5 4 7 0
```

Output

```
9
0 2 1 3
```

Giải thích ví dụ

Lộ trình Thiên Mệnh là đi từ **Điểm xuất phát (0)** -> **Trận Nhãn 2** -> **Trận Nhãn 1** -> **Trận Nhãn 3**. Tổng chi phí = dist[0][2] + dist[2][1] + dist[1][3] = 2 + 3 + 4 = 9.

Mục tiêu: Từ ma trận tri thức, tìm lộ trình đi qua tất cả các điểm đúng một lần với tổng chi phí nhỏ nhất. Đây là biến thể của bài toán Người du lịch (TSP).

- **Vấn đề:** Nếu dùng thuật toán quay lui (Backtracking) thông thường, độ phức tạp là O(N!). Với N=15, 15! ≈ 1.3 x10^{12} phép tính, máy tính sẽ không thể xử lý nổi trong 1 giây.
- **Cấu trúc dữ liệu tối ưu: Bitmask DP (Quy hoạch động trạng thái Bit)**
  - **Bitmask (Số nguyên):** Sử dụng một số nguyên int mask để biểu diễn tập hợp các Trận Nhãn đã đi qua.
    - Ví dụ: N=3. mask = 5 (nhị phân 101) nghĩa là đã đi qua Trận Nhãn thứ 0 và thứ 2, chưa qua thứ 1.
  - **Bảng phương án (int dp[1 << N][N]):**
    - dp[mask][i]: Chi phí nhỏ nhất để đi qua tập hợp các điểm trong mask và kết thúc tại điểm i.
    - Kích thước bảng: 2^{15} x 15 ≈ 32768 x 15 ≈ 491,520 phần tử (Rất nhỏ so với bộ nhớ 512MB).
    - Giải thuật:

dp[mask | (1<< v)][v] = min(dp[mask | (1 << v)][v], dp[mask][u] + cost[u][v])

- **Đánh giá:** Độ phức tạp giảm xuống còn O(2^N . N^2), cho phép "Thiên Cơ" tìm ra đáp án tối ưu (S) gần như tức thời.

3.2.3 Giải quyết Bài toán 3: Nhập Thế - Hiện thực hóa Game

Giới hạn thời gian: **1.0 giây**

Giới hạn bộ nhớ: **512 MB**

Giới thiệu

Giai đoạn cuối cùng của sự giác ngộ là "Nhập Thế" - hành động trong thế giới thực. Lý thuyết và kế hoạch giờ đây phải được kiểm chứng bằng bản lĩnh. Kỳ Môn Sư sẽ trực tiếp đặt chân lên Thiên Không Trận Đồ, không còn dựa vào ma trận tri thức có sẵn mà phải tự mình cảm nhận và tính toán. Đây là thử thách tối thượng, tổng hợp toàn bộ trí tuệ từ hai giai đoạn trước. Bạn phải đối mặt với **Thiên Cơ** - ý thức của trận đồ - và tìm ra con đường hoàn hảo nhất để tái lập lại sự cân bằng.

Đề bài

- Cho một Thiên Không Trận Đồ dưới dạng lưới R x C, bao gồm các ô đất trống ('.'), núi non hiểm trở ('#'), và N Trận Nhãn ('S'). Kỳ Môn Sư bắt đầu hành trình tại ô (1, 1). Từ một ô, họ có thể di chuyển đến các ô kề cạnh (trên, dưới, trái, phải), miễn là ô đó không phải là núi.
- Nhiệm vụ của bạn là tìm ra tổng số bước đi tối thiểu để Kỳ Môn Sư có thể bắt đầu từ ô (1, 1) và khai mở tất cả N Trận Nhãn trên trận đồ.

Input

- Dòng đầu tiên chứa ba số nguyên dương R, C, và N ($1 \leq R, C \leq 100$, $1 \leq N \leq 15$).
- R dòng tiếp theo, mỗi dòng chứa C ký tự, mô tả trận đồ.
- Dữ liệu đảm bảo có đúng N ký tự 'S', ô (1, 1) luôn là một ô đất trống.

Output

Nếu không tồn tại lộ trình thỏa mãn, in ra -1. Ngược lại, in ra hai dòng:

- Dòng đầu tiên: một số nguyên là tổng số bước đi tối thiểu của lộ trình Thiên Mệnh.
- Dòng thứ hai: N + 1 số nguyên, là thứ tự các địa điểm được khai mở trong lộ trình tối ưu, bắt đầu bằng 0.

Ví dụ

Input

```
3 6 3
.S..#.
..#.#.
S.#S..
```

Output

```
9
0 2 1 3
```

Giải thích ví dụ

- Điểm xuất phát 0: (1, 1). Trận Nhãn 1: (1, 2). Trận Nhãn 2: (3, 1). Trận Nhãn 3: (3, 4).

- Lộ trình Thiên Mệnh là đi từ **Điểm xuất phát (0)** -> **Trận Nhãn 2** -> **Trận Nhãn 1** -> **Trận Nhãn 3**. Tổng chi phí = dist[0][2] + dist[2][1] + dist[1][3] = 2 + 3 + 4 = 9.

Mục tiêu: Xây dựng môi trường tương tác thời gian thực, nơi người chơi tự mình giải đố dựa trên kết quả của bài toán 2.

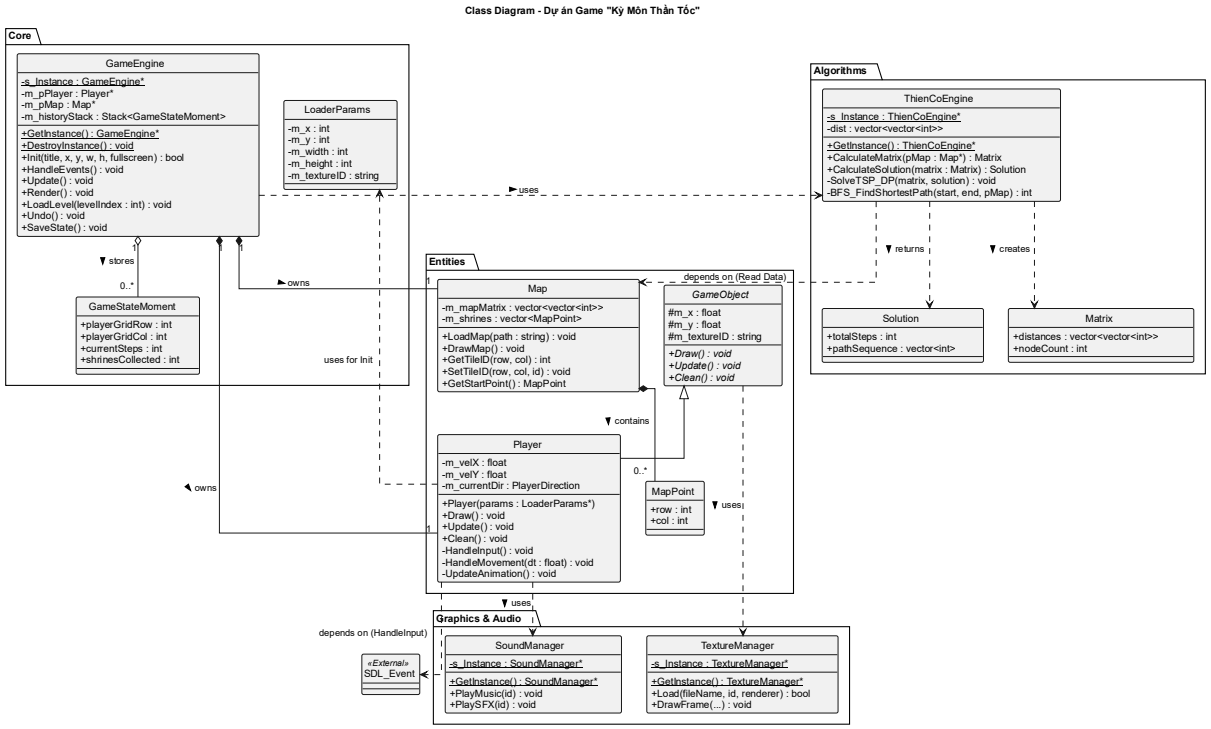
- **Cấu trúc dữ liệu cho Game Engine:**
  - **Lớp Map (OOP):** Lưu trữ lưới m_mapMatrix và danh sách tọa độ vector<Point> m_shrines.
  - **Stack Hoàn tác (std::stack<GameState>):** Để thực hiện chức năng "Hồi Tưởng" (Undo).
    - Mỗi khi người chơi di chuyển, một bản sao trạng thái (vị trí nhân vật, các trận nhãn đã ăn, số bước đi) được đẩy (push) vào Stack.
    - Khi bấm 'U', hệ thống lấy (pop) trạng thái trên cùng ra và khôi phục lại. Cấu trúc LIFO (Last In, First Out) của Stack là hoàn hảo cho tính năng này.
  - **Bộ nhớ đệm đồ họa (Texture Cache):** Sử dụng std::map<string, SDL_Texture*> để quản lý tài nguyên hình ảnh, tránh việc nạp lại cùng một ảnh nhiều lần gây tốn RAM.

PHẦN 4: PHÂN TÍCH HƯỚNG ĐỐI TƯỢNG VÀ TRIỂN KHAI HỆ THỐNG

4.1. Cấu trúc hệ thống hướng đối tượng (Class, Object, Relation)

Giai đoạn "Nhập Thế" là bước chuyển mình từ tư duy thuật toán sang kiến trúc phần mềm. Hệ thống được thiết kế dựa trên tư duy **Lập trình hướng đối tượng (OOP)** kết hợp với mẫu thiết kế **Singleton Pattern** (Mẫu đơn lập) để quản lý tài nguyên và trạng thái trò chơi một cách tập trung, chặt chẽ.

4.1.1. Sơ đồ lớp *(Class Diagram)*



Hình 3: Sơ đồ lớp (Class Diagram)

Hệ thống được chia thành 3 khối chính tương tác chặt chẽ: **Core (Lõi hệ thống)**, **Entities (Thực thể game)**, và **Algorithms (Thuật toán trí tuệ)**.

4.1.2. Các thành phần chi tiết và vai trò

a) Khối Core (Lõi hệ thống)

- **Class GameEngine (Singleton Context):**
  - **Vai trò:** Là lớp trung tâm ("Bộ não điều phối"), chịu trách nhiệm khởi tạo thư viện SDL (Init), duy trì vòng lặp game vô tận (Game Loop), và xử lý chuyển đổi trạng thái (FSM).

- **Dữ liệu:** Quản lý các con trỏ quan trọng nhất: SDL_Window*, SDL_Renderer*, con trỏ đến Player và Map. Đặc biệt chứa std::stack<GameStateMoment> để phục vụ tính năng Undo.
- **Hành vi:** Điều phối luồng chương trình thông qua 3 hàm chủ đạo: HandleEvents() (Nhận lệnh), Update() (Xử lý logic), Render() (Vẽ hình).

- **Class ResourceManagers (TextureManager & SoundManager):**
  - **Vai trò:** "Nhà kho" thông minh. Quản lý việc nạp/giải phóng hình ảnh và âm thanh.
  - **Đặc điểm:** Sử dụng cấu trúc std::map để ánh xạ ID (chuỗi ký tự) sang địa chỉ bộ nhớ. Đảm bảo mỗi tài nguyên chỉ được nạp một lần duy nhất (Lazy Loading), tránh rò rỉ bộ nhớ.

b) Khối Entities (Thực thể - Áp dụng Tính Kế Thừa & Đa Hình)

Đây là nơi áp dụng mạnh mẽ nguyên lý Inheritance và Polymorphism.

- **Abstract Class GameObject:**
  - **Vai trò:** Lớp cơ sở trừu tượng định nghĩa "khuôn mẫu" cho mọi vật thể trong game (Nhân vật, NPC, Vật cản).
  - **Phương thức thuần ảo:**
    - Draw(): Vẽ đối tượng lên màn hình.
    - Update(): Cập nhật trạng thái vật lý.
    - Clean(): Dọn dẹp tài nguyên.
- **Concrete Classes (Các lớp con hiện thực hóa):**
  - **Class Player (Kỳ Môn Sư):** Kế thừa từ GameObject.
    - *Vai trò:* Xử lý logic di chuyển nội suy (Interpolation) để trượt mượt mà trên lưới. Chứa máy trạng thái nội bộ (State Machine) cho Animation: Idle, Run, Falling.
  - **Class Map (Thiên Không Trận Đồ):**
    - *Vai trò:* Bao đóng (Encapsulate) ma trận dữ liệu bản đồ. Chịu trách nhiệm chuyển đổi từ file .txt sang các đối tượng đồ họa (Tilemap) và tự động căn chỉnh tỷ lệ (Auto-scaling).

c) Khối Algorithms (Trí tuệ nhân tạo - Áp dụng Tính Đóng Gói)

- **Class ThienCoEngine:**

- **Vai trò:** Một phiên bản thu nhỏ của logic game dùng để chạy mô phỏng tự động.
- **Hành vi:** Che giấu hoàn toàn sự phức tạp của thuật toán BFS và Bitmask DP. Các lớp bên ngoài chỉ cần gọi CalculateSolution() để nhận về kết quả tối ưu (Rank S) mà không cần biết cách thức vận hành bên trong.

4.1.3. Các quan hệ giữa các lớp (Relationships)

a) Quan hệ Composition (Hợp thành - "Has-a"):

- GameEngine sở hữu Player và Map. (Nếu GameEngine hủy, toàn bộ thế giới game và nhân vật sẽ hủy theo).
- Map sở hữu danh sách các Shrine (Trận Nhãn).

b) Quan hệ Inheritance (Kế thừa - "Is-a"):

- Player là một GameObject. Điều này giúp GameEngine có thể quản lý đối tượng thông qua con trỏ lớp cha GameObject*.

c) Quan hệ Association (Kết hợp - "Uses"):

- Player sử dụng Map để kiểm tra va chạm (Collision Detection).
- GameEngine sử dụng ThienCoEngine để lấy dữ liệu so sánh kết quả.
- Tất cả các lớp sử dụng TextureManager để vẽ hình ảnh.

4.1.4. Áp dụng các nguyên lý OOP trong dự án

- **Tính Đóng Gói (Encapsulation):**
  - Class Map che giấu ma trận m_mapMatrix (private). Các lớp khác (như Player) muốn biết thông tin ô đất phải dùng phương thức GetTileID(row, col). Điều này bảo vệ dữ liệu bản đồ khỏi bị thay đổi sai lệch từ bên ngoài.
  - Class ThienCoEngine đóng gói toàn bộ thuật toán tìm đường phức tạp, chỉ lộ ra giao diện đơn giản là CalculateSolution.
- **Tính Kế Thừa (Inheritance):**
  - Việc tách GameObject làm lớp cha giúp tái sử dụng mã nguồn cho các thuộc tính chung như tọa độ (x, y), kích thước (width, height) và ID hình ảnh.
- **Tính Đa Hình (Polymorphism):**
  - Phương thức Update() hoạt động khác nhau ở mỗi đối tượng:

- Ở Player: Tính toán vận tốc, hướng di chuyển và animation.
- Ở Map: Tính toán vị trí hiển thị (Offset) để luôn nằm giữa màn hình.

o Điều này cho phép vòng lặp chính của game gọi Update() một cách đồng nhất mà không cần kiểm tra loại đối tượng.

4.2. Triển khai thuật toán và Vòng lặp chính

4.2.1. Vòng lặp Game (The Game Loop)

Trái tim của GameEngine nằm ở phương thức Run() (được cài đặt trong main.cpp và GameEngine::Update), nơi duy trì sự sống của trò chơi:

```
// Minh họa mã giả Vòng lặp chính
// FRAME_DELAY = 1000 / FPS (FPS = 60)
void GameEngine::Run() {
    while (m_bRunning) {
        frameStart = SDL_GetTicks();
        HandleEvents(); // 1. Tiếp nhận thao tác người chơi (Phím bấm)
        Update();       // 2. Cập nhật logic game (Vật lý, AI, Va chạm)
        Render();       // 3. Vẽ lại toàn bộ thế giới ra màn hình
        // Cơ chế ổn định khung hình (Frame Capping)
        frameTime = SDL_GetTicks() - frameStart;
        if (FRAME_DELAY > frameTime) {
            SDL_Delay(FRAME_DELAY - frameTime);
        }
    }
}
```

4.2.2. Cơ chế máy trạng thái (Finite State Machine)

Dù không tách ra thành các class State riêng biệt (như MenuState, PlayState) để tránh quá phức tạp cho đồ án quy mô nhỏ, chúng em sử dụng enum GameState kết hợp với Switch-Case để quản lý hành vi.

- Tùy thuộc vào m_currentState đang là STATE_PLAY hay STATE_INTRO, hành vi thực thi trong Update() và Render() sẽ khác nhau hoàn toàn.
- **Điểm mạnh:** Đơn giản, hiệu quả, dễ debug và tích hợp tốt với hiệu ứng chuyển cảnh (Fade Transition).

4.2.3. Triển khai AI Thiên Cơ

Logic AI được chạy trên một luồng tách biệt (hoặc chạy trước khi vào màn chơi) để không làm gián đoạn vòng lặp đồ họa.

- **Bước 1:** ThienCoEngine nhận con trỏ Map.
- **Bước 2:** Chạy BFS để xây dựng đồ thị khoảng cách.
- **Bước 3:** Chạy thuật toán **Quy hoạch động với Bitmask (DP Bitmask)** để giải quyết bài toán Người Du Lịch (TSP) từ ma trận khoảng cách, tìm ra chu trình ngắn nhất đi qua tất cả các trận nhãn.
- **Bước 4:** Trả về kết quả optimalSteps cho GameEngine lưu trữ.

PHẦN 5: KẾT QUẢ CHẠY CHƯƠNG TRÌNH

5.1. Giao diện chung

5.2. Giao diện chi tiết

5.2.1. Trang chủ (Home)

Đây là điểm chạm đầu tiên của người chơi với hệ thống. Màn hình chờ được thiết kế tối giản nhưng ấn tượng, với hình ảnh Rồng thần và Kỳ Môn Sư đứng giữa biển mây, thể hiện khí thế sẵn sàng "Nhập Thế".

Các chức năng điều hướng được bố trí gọn gàng phía dưới màn hình, sử dụng các phím tắt quen thuộc để tối ưu trải nghiệm người dùng (UX):

- [ENTER]: Bắt đầu hành trình (Start).
- [1] HOME: Quay về màn hình chính.
- [2] MUSIC: Bật/Tắt âm thanh để tăng sự tập trung.
- [U] UNDO: Kích hoạt tính năng Hồi tưởng.
- [Q] QUIT: Thoát chương trình.
- [F11] FULL SCREEN: Chuyển đổi chế độ toàn màn hình.

Hình 4: Trang chủ (Home) của chương trình

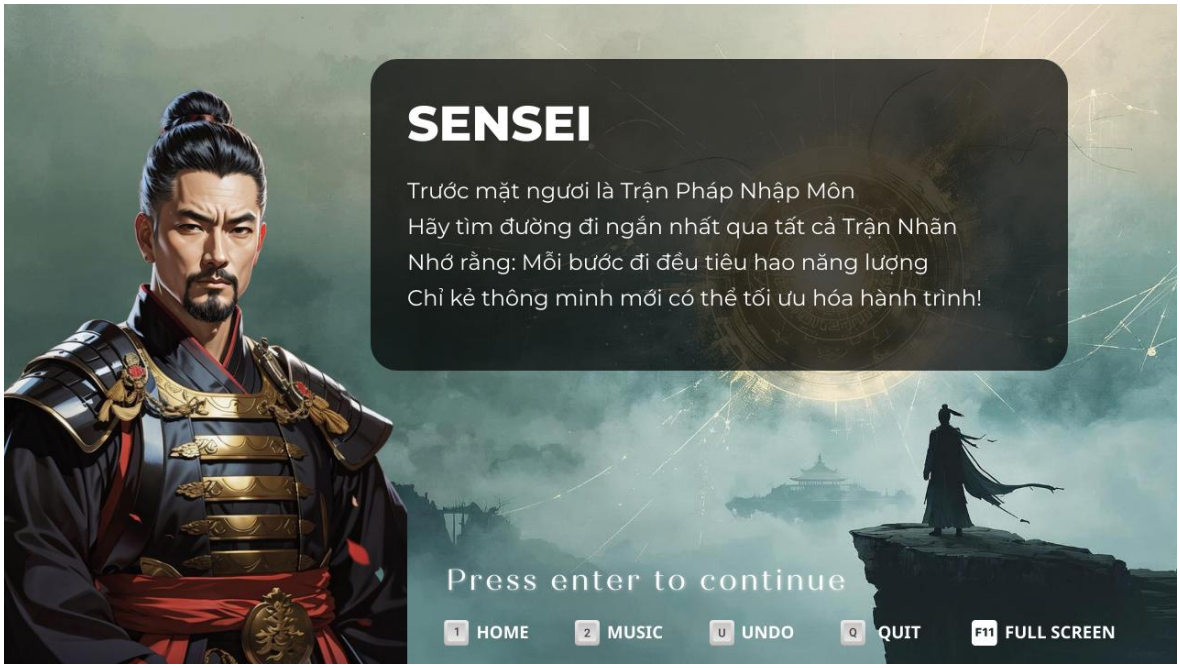
5.2.2. Các vòng chơi

Hệ thống màn chơi trong "Kỳ Môn Thần Tốc" được thiết kế theo cấu trúc tuyến tính, đưa người chơi đi từ trạng thái "Nhập Môn" đến "Giác Ngộ". Trước mỗi vòng chơi, game không đưa người dùng vào mê cung ngay lập tức mà dẫn dắt qua một lớp màn hình chuyển tiếp (Transition Screen) và lời giáo huấn của Sư Phụ (Sensei).

a) Vòng 1: Thiên Cơ Viện (The Awakening)

- **Bối cảnh:** Đây là nơi khởi đầu của mọi Kỳ Môn Sư. Khung cảnh là những đỉnh núi thấp thoáng trong mây trắng, gợi mở sự khởi đầu nhẹ nhàng.
- **Lời dạy của Sensei:** *"Hành trình vạn dặm bắt đầu từ một bước chân."* Người chơi được làm quen với Trận Pháp Nhập Môn.
- **Đặc điểm kỹ thuật:**
  - Kích thước bản đồ nhỏ (ví dụ 5x5 hoặc 10x10).
  - Số lượng Trận Nhãn ít (N = 5).
  - Ít vật cản.
- **Mục tiêu:** Giúp người chơi làm quen với cơ chế di chuyển và khái niệm "tiết kiệm bước đi". Tại đây, thuật toán AI chạy rất nhanh và kết quả tối ưu thường dễ dàng được người chơi nhận ra bằng mắt thường.

Hình 5: Giao diện vòng 1 - Thiên cơ viện

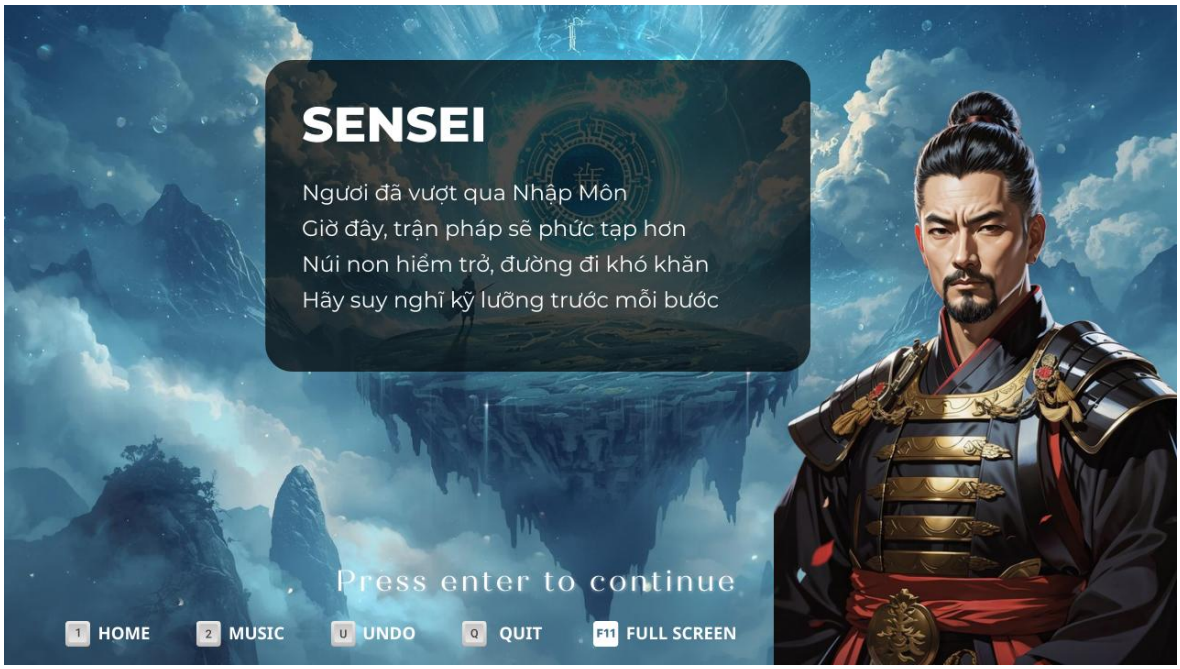


Hình 6: Lời dặn và hướng dẫn của Sensei ở vòng 1

b) Vòng 2: Thử Thách (The Trial)

- **Bối cảnh:** Màu sắc chuyển sang tông xanh đậm hơn, thể hiện sự nghiêm túc và nguy hiểm. Những ngọn núi trở nên sắc nhọn, hiểm trở hơn.
- **Lời dạy của Sensei:** *"Núi non hiểm trở, đường đi khó khăn. Hãy suy nghĩ kỹ lưỡng trước mỗi bước."*
- **Đặc điểm kỹ thuật:**
  - Kích thước bản đồ mở rộng.
  - Xuất hiện nhiều vật cản địa hình (Tường #, Vực thẳm).
  - Thuật toán **BFS** bắt đầu phát huy tác dụng rõ rệt khi đường chim bay không còn là đường ngắn nhất. Người chơi buộc phải tính toán chi phí di chuyển vòng qua các chướng ngại vật.

Hình 7: Giao diện vòng 2 – Thử thách

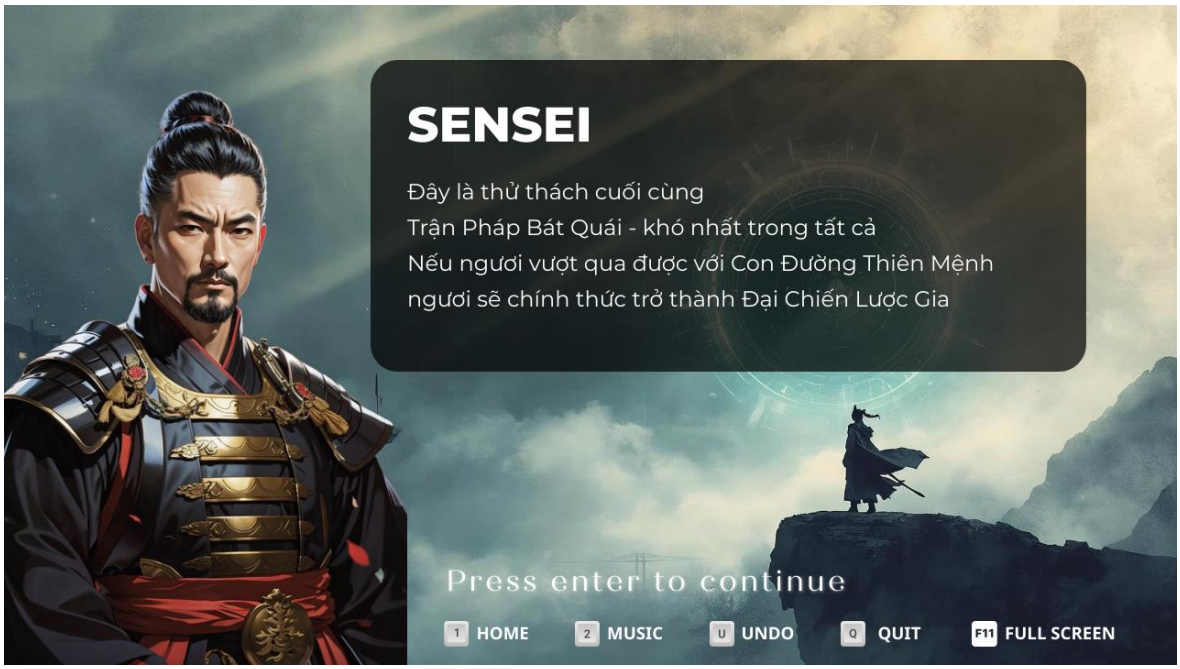


Hình 8: Lời dặn và hướng dẫn của Sensei ở vòng 2

c) Vòng 3: Trận Pháp Bát Quái (The Grand Master)

- **Bối cảnh:** Đỉnh cao của nghệ thuật chiến thuật. Ánh sáng vàng kim từ những chiếc đèn lồng đối lập với nền trời xanh thẫm, tạo nên không khí trang nghiêm của một trận chiến cuối cùng.
- **Lời dạy của Sensei:** *"Nếu ngươi vượt qua được với Con Đường Thiên Mệnh, ngươi sẽ chính thức trở thành Đại Chiến Lược Gia."*
- **Đặc điểm kỹ thuật:**
  - Đây là bài kiểm tra tổng hợp (Final Exam) cho cả người chơi và hệ thống.
  - Số lượng Trận Nhãn đạt mức tối đa (N = 15).
  - Cấu trúc mê cung phức tạp như Bát Quái Trận.
  - **Thách thức thuật toán:** Đây là lúc thuật toán **Bitmask DP** tỏa sáng. Với số lượng điểm lớn, con người rất khó tìm ra đường đi ngắn nhất bằng trực giác, nhưng AI "Thiên Cơ" vẫn giải quyết trong tích tắc. Sự chênh lệch giữa Rank S (Máy) và Rank A (Người) thường rất lớn ở vòng này, tạo động lực để người chơi sử dụng chức năng **Undo** và thử lại nhiều lần.

Hình 9: Giao diện vòng 3 – Trận pháp bát quái



Hình 10: Lời dặn và hướng dẫn của Sensei ở vòng 3

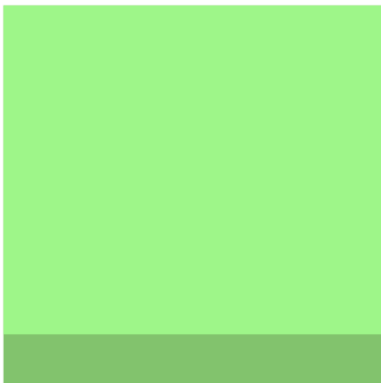
5.2.3 Giao diện chơi (In-Game Interface)

Hình 11: Giao diện trong game

a) Thiên Không Trận Đồ (The Grid Map)

Điểm nhấn trung tâm của màn hình là hệ thống lưới (Grid System) đại diện cho trận pháp:

- **Các ô địa hình (Tiles):** Sử dụng tông màu xanh lá mạ và xanh thiên thanh xen kẽ, tạo cảm giác như những mảnh đất đang lơ lửng giữa tầng không. Đây là các ô an toàn mà nhân vật có thể di chuyển.



Hình 12: Ô an toàn có thể di chuyển

- **Chướng ngại vật (Obstacles):** Các ô chứa tảng đá màu xám bạc đại diện cho "Núi non hiểm trở". Đây là các vùng cấm địa, buộc người chơi phải tìm đường vòng, qua đó kích thích tư duy tìm kiếm giải pháp tối ưu.

Hình 13: Hòn đá gây cản trở bước đi của nhân vật

- **Trận Nhãn (Objectives):** Được biểu thị bằng các biểu tượng tròn mang họa tiết **Âm Dương** rực rỡ sắc vàng cam. Đây là các điểm mục tiêu cần "Khai Mở".

Hình 14: Trận nhãn – Mục tiêu cần "Khai mở"

b) Nhân vật và Hiệu ứng

- **Kỳ Môn Sư (Player):** Được thiết kế dưới dạng Pixel Art nhỏ gọn, đứng nổi bật trên nền gạch

Hình 15: Nhân vật trong game – Kỳ Môn Sư

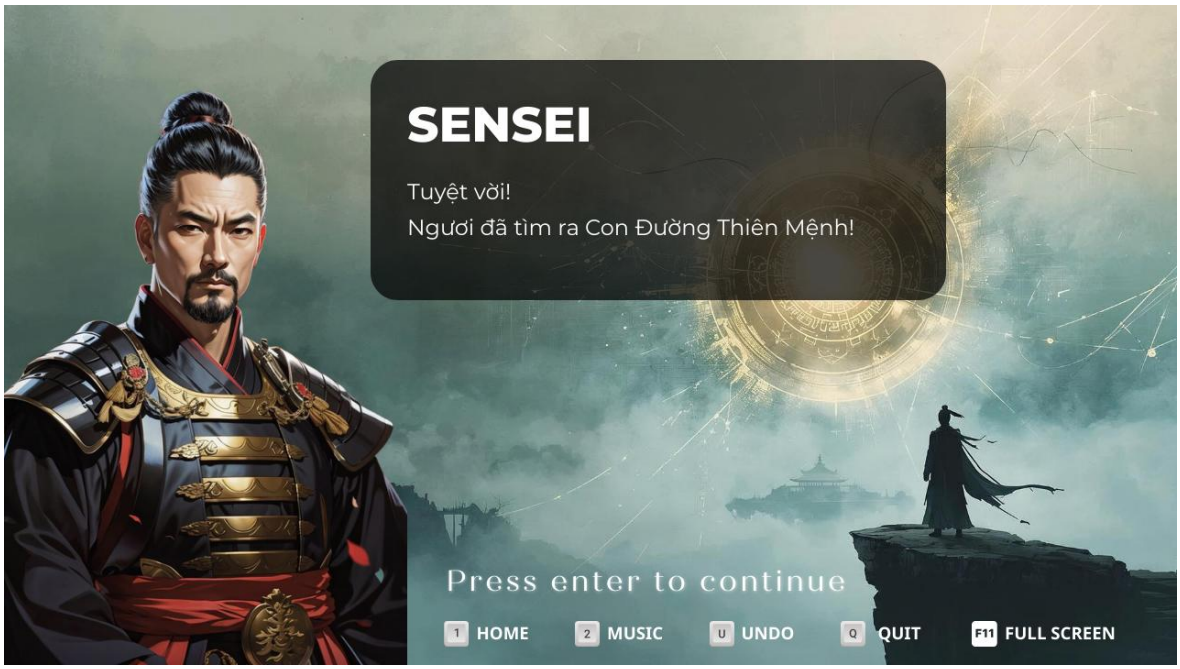
- **Nền (Background):** Thay vì một màu đen đơn điệu, chúng em sử dụng hình ảnh bầu trời với những đám mây ngũ sắc trôi nhẹ, tạo chiều sâu không gian và duy trì cảm giác thư thái, giúp giảm áp lực cho người chơi khi phải suy nghĩ lâu.

Hình 16: Hình nền xuyên suốt trò chơi

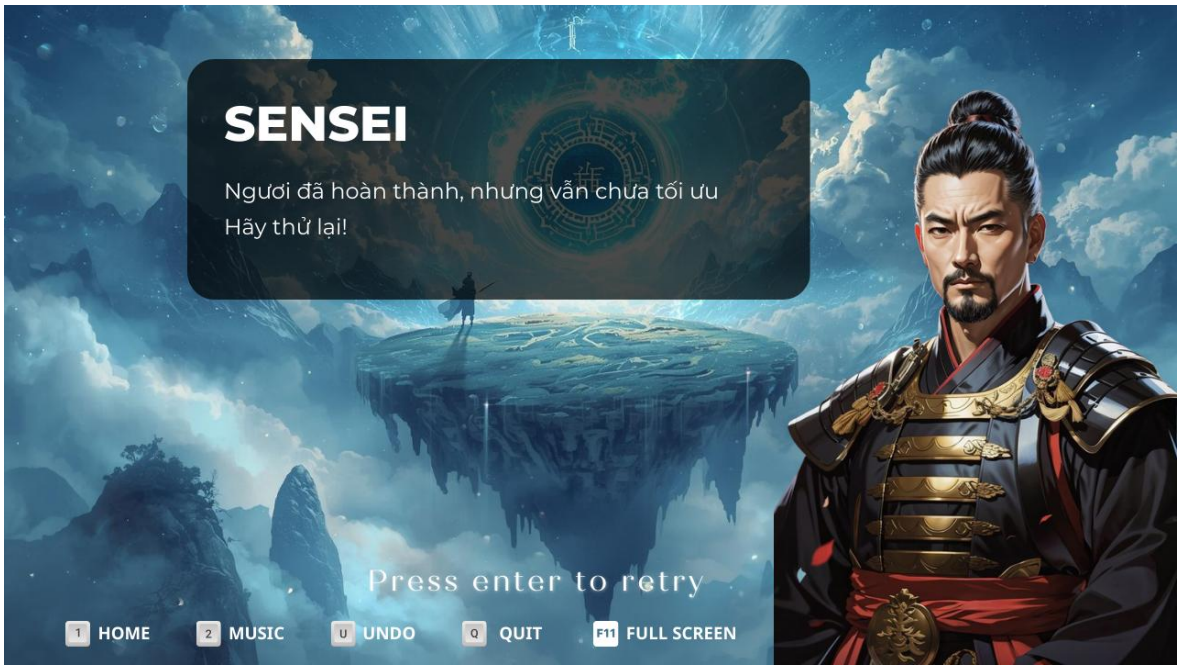
c) Thanh điều khiển (Control Bar) Phía dưới cùng màn hình là thanh công cụ trong suốt, hiển thị nhắc nhở về các phím chức năng quan trọng (HOME, MUSIC, UNDO...). Việc bố trí này giúp giao diện trở nên gọn gàng, không che khuất tầm nhìn vào bản đồ, đồng thời hỗ trợ người chơi mới làm quen với thao tác điều khiển mà không cần vào menu hướng dẫn.

Hình 17: Thanh điều khiển

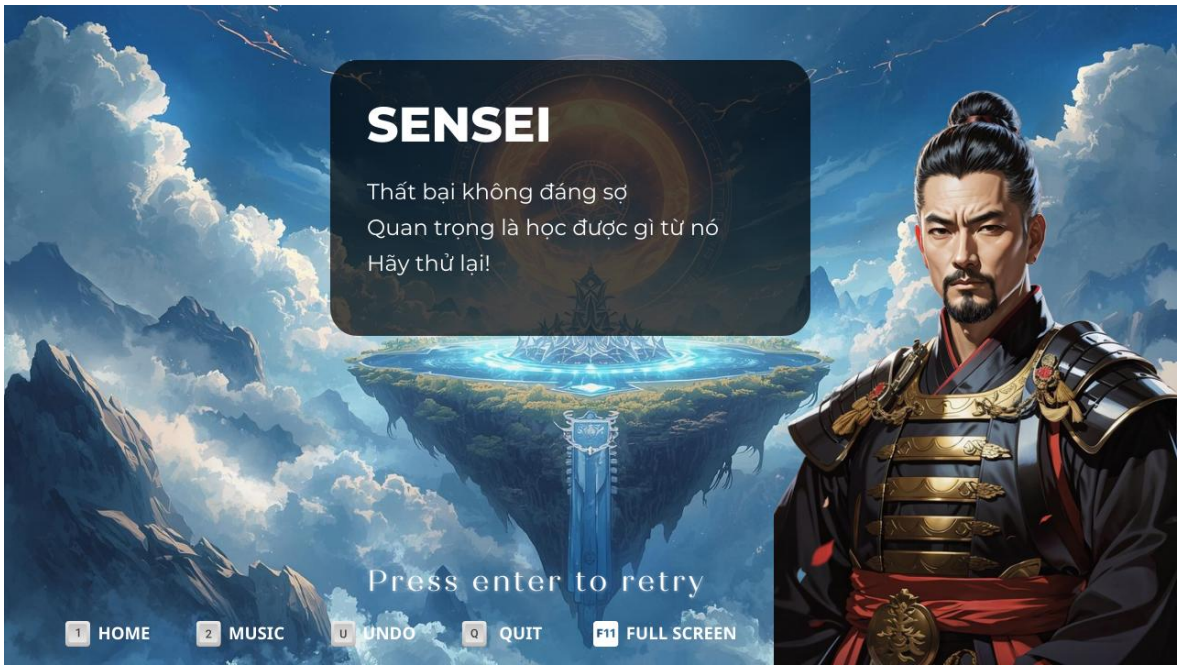
5.2.4. Lời nhận xét của Sư Phụ (Sensei) sau mỗi màn chơi



Hình 18: Lời nhận xét khi vượt qua màn chơi



Hình 19: Lời nhận xét khi bước đi chưa tối ưu



Hình 20: Lời khuyên khi chơi thất bại

5.2.5. Kết thúc trò chơi

Hình 21: Giao diện kết thúc trò chơi

PHẦN 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1. Đánh giá kết quả thực hiện (Product Assessment)

Sau quá trình nghiên cứu, thiết kế và thi công mã nguồn, đồ án "Kỳ Môn Thần Tốc" đã hoàn thành và đáp ứng được các mục tiêu đề ra ban đầu. Sản phẩm không chỉ là một trò chơi giải trí mà là một hệ thống phần mềm hoàn chỉnh, minh chứng cho sự kết hợp giữa thuật toán tối ưu và kỹ thuật lập trình ứng dụng.

6.1.1. Những điểm đạt được (Ưu điểm)

- **Về mặt Thuật toán (Academic Value):**
  - Đã hiện thực hóa thành công "bộ não" **Thiên Cơ (AI)** dựa trên sự kết hợp giữa **BFS** (tìm khoảng cách thực tế trên địa hình phức tạp) và **Quy hoạch động Bitmask** (giải quyết bài toán TSP). Đây là điểm sáng lớn nhất về mặt học thuật, cho phép hệ thống đưa ra lời giải chính xác tuyệt đối (Rank S) trong thời gian thực (< 1 giây với N <= 15).
- **Về mặt Kỹ thuật phần mềm (Software Engineering):**
  - Kiến trúc hệ thống được xây dựng bài bản theo tư duy **Hướng đối tượng (OOP)**. Việc áp dụng các mẫu thiết kế như **Singleton** (cho GameEngine, Managers) và cơ chế quản lý tài nguyên tự động giúp mã nguồn trong sáng, dễ bảo trì và hạn chế tối đa rò rỉ bộ nhớ.
  - Hệ thống **Game Loop** hoạt động ổn định, duy trì tốc độ khung hình (FPS) mượt mà nhờ cơ chế kiểm soát thời gian (Delta Time).
- **Về mặt Trải nghiệm người dùng (UX):**
  - Tính năng **Auto-scaling** hoạt động hiệu quả, giải quyết triệt để vấn đề hiển thị bản đồ kích thước khác nhau trên cùng một cửa sổ.
  - Cơ chế **Undo (Hoàn tác)** dựa trên cấu trúc Stack hoạt động chính xác, cho phép người chơi sửa sai mà không gặp lỗi logic hay mất dữ liệu.

6.1.2. Những điểm hạn chế (Tồn tại)

Bên cạnh những ưu điểm, với giới hạn về thời gian và kiến thức của sinh viên năm 2, đồ án vẫn còn một số điểm cần cải thiện:

- **Sự đa dạng của màn chơi:** Hiện tại, dữ liệu bản đồ phụ thuộc hoàn toàn vào các file .txt được tạo thủ công. Việc chưa có một công cụ biên tập bản

đồ trực quan (Map Editor) khiến việc tạo ra các màn chơi mới tốn nhiều thời gian.

- **Tính động của môi trường:** AI hiện tại chỉ giải quyết bài toán trên bản đồ tĩnh. Trò chơi chưa có các chướng ngại vật di động (Dynamic Obstacles) hoặc kẻ thù đuổi theo người chơi, làm giảm đi tính kịch tính ở các màn cấp cao.
- **Hiệu ứng đồ họa:** Dù đã có Texture và Animation, nhưng các hiệu ứng cháy nổ, hạt (Particles) hay ánh sáng (Lighting) vẫn còn sơ sài, chưa khai thác hết sức mạnh phần cứng đồ họa.

6.2. Hướng phát triển (Future Development)

Để "Kỳ Môn Thần Tốc" phát triển từ một đồ án môn học thành một sản phẩm có thể phát hành thực tế, nhóm đề xuất các hướng nâng cấp sau:

a) Xây dựng Map Editor (Trình biên tập bản đồ):

- Phát triển một công cụ GUI tích hợp ngay trong game, cho phép người dùng tự vẽ địa hình, đặt Trận Nhãn và lưu lại dưới dạng file level. Điều này sẽ mở ra khả năng "User Generated Content" (Nội dung do người dùng tạo), kéo dài tuổi thọ của game.

b) Nâng cấp Thuật toán tìm đường:

- Đối với các bản đồ kích thước rất lớn, thuật toán BFS có thể trở nên chậm chạp. Hướng phát triển là tích hợp thuật toán **A* (A-Star)** hoặc **JPS (Jump Point Search)** để tối ưu hóa hiệu năng tìm đường.
- Nghiên cứu áp dụng thuật toán **Heuristic** cho bài toán TSP khi số lượng Trận Nhãn $N > 20$, chấp nhận kết quả "gần tối ưu" để đổi lấy tốc độ xử lý.

c) Mở rộng Gameplay:

- Thêm các loại ô gạch chức năng: Ô dịch chuyển tức thời (Teleport), Ô băng trượt (Ice Tile), Cửa và Chìa khóa.
- Thêm chế độ "Đối kháng": Hai người chơi cùng giải một bản đồ, ai xong trước và ít bước hơn sẽ thắng.

d) Đa nền tảng (Cross-platform):

- Tận dụng khả năng của SDL2 để đóng gói game lên nền tảng di động (Android/iOS), biến "Kỳ Môn Thần Tốc" thành một game mobile giải đố tiện lợi.

TÀI LIỆU THAM KHẢO

Ngoài việc vận dụng các kiến thức về Cấu trúc & Giải thuật, Lập trình hướng đối tượng (OOP) và các môn học liên quan khác ở trường, chúng em đã tìm hiểu chuyên sâu về các thuật toán, thiết kế thuật toán, cách viết chương trình và giao diện trên SDL2 trên các nền tảng khác.

1. Thuật toán Tìm kiếm theo chiều rộng (Breadth-First Search - BFS)

2. Thuật toán Quy hoạch động trạng thái Bitmask (Bitmask Dynamic Programming)

3. Lập trình hướng đối tượng nâng cao

4. Giao diện SDL2

PHỤ LỤC

Cấu trúc thư mục:

```
GameKyMonThanToc:
│   .gitignore
│   CMakeLists.txt
│   README.md
│
├──.vscode
│       c_cpp_properties.json
│       launch.json
│
├──assets
│   ├──audio
│   │   ├──bgm
│   │   │       bgm_game.mp3
│   │   │       bgm_menu.mp3
│   │   │
│   │   └──sfx
│   │           sfx_click.wav
│   │           sfx_collect.wav
│   │           sfx_lose.wav
│   │           sfx_step.wav
│   │           sfx_win.wav
│   │
│   ├──fonts
│   │       Roboto-Regular.ttf
│   │
│   ├──images
│   │   ├──characters
│   │   │       idle_down.png
│   │   │       idle_left.png
│   │   │       idle_right.png
│   │   │       idle_up.png
│   │   │       run_down.png
│   │   │       run_left.png
│   │   │       run_right.png
│   │   │       run_up.png
│   │   │
│   │   └──environment
│   │           bg_parallax.png
│   │           img_continue.png
│   │           img_intro.png
│   │           img_lose.png
│   │           img_scence1_name.png
│   │           img_scence1_rankA.png
│   │           img_scence1_rankS.png
│   │           img_scence1_sensei.png
│   │           img_scence2_name.png
│   │           img_scence2_rankA.png
│   │           img_scence2_rankS.png
│   │           img_scence2_sensei.png
│   │           img_scence3_name.png
│   │           img_scence3_rankA.png
│   │           img_scence3_rankS.png
│   │           img_scence3_sensei.png
```

```
│   │           tileset.png
│   │
│   └──levels
│           level1.txt
│           level10.txt
│           level11.txt
│           level12.txt
│           level13.txt
│           level14.txt
│           level15.txt
│           level16.txt
│           level17.txt
│           level18.txt
│           level19.txt
│           level2.txt
│           level20.txt
│           level21.txt
│           level22.txt
│           level3.txt
│           level4.txt
│           level5.txt
│           level6.txt
│           level7.txt
│           level8.txt
│           level9.txt
│
├──build
│   │   KyMonThanToc.exe
│   │
├──docs
│   │   BFS.svg
│   │   Class_Diagram.svg
│   │   DP_Bitmask.svg
│   │   Ke Hoạch PBL_ 2_Dự án CSLT_2025_2026.pdf
│   │   MauBaoCao_PBL2.docx.pdf
│   │
│   ├──Bai_1_Minh_Tri
│   │   │   Bai1.cpp
│   │   │   Bài toán 1 Minh Trí - Khai Mở Tri Thức Trận Đồ.docx
│   │   │
│   │   └──tests
│   │       ├──input
│   │       │       input00.txt
│   │       │       input01.txt
│   │       │       input02.txt
│   │       │       input03.txt
│   │       │       input04.txt
│   │       │       input05.txt
│   │       │       input06.txt
│   │       │       input07.txt
│   │       │       input08.txt
│   │       │       input09.txt
│   │       │       input10.txt
│   │       │       input11.txt
│   │       │       input12.txt
│   │       │       input13.txt
```

```
│   │       │       input14.txt
│   │       │       input15.txt
│   │       │       input16.txt
│   │       │       input17.txt
│   │       │       input18.txt
│   │       │       input19.txt
│   │       │       input20.txt
│   │       │
│   │       └───output
│   │               output00.txt
│   │               output01.txt
│   │               output02.txt
│   │               output03.txt
│   │               output04.txt
│   │               output05.txt
│   │               output06.txt
│   │               output07.txt
│   │               output08.txt
│   │               output09.txt
│   │               output10.txt
│   │               output11.txt
│   │               output12.txt
│   │               output13.txt
│   │               output14.txt
│   │               output15.txt
│   │               output16.txt
│   │               output17.txt
│   │               output18.txt
│   │               output19.txt
│   │               output20.txt
│   │
│   ├───Bai_2_Dinh_Menh
│   │   │   Bai2.cpp
│   │   │   Bài toán 2 Định Mệnh - Vạch Ra Thiên Mệnh.docx
│   │   │
│   │   └───tests
│   │       ├───input
│   │       │       input00.txt
│   │       │       input01.txt
│   │       │       input02.txt
│   │       │       input03.txt
│   │       │       input04.txt
│   │       │       input05.txt
│   │       │       input06.txt
│   │       │       input07.txt
│   │       │       input08.txt
│   │       │       input09.txt
│   │       │       input10.txt
│   │       │       input11.txt
│   │       │       input12.txt
│   │       │       input13.txt
│   │       │       input14.txt
│   │       │       input15.txt
│   │       │       input16.txt
│   │       │       input17.txt
│   │       │       input18.txt
```

```
│   │       │       input19.txt
│   │       │       input20.txt
│   │       │
│   │       └───output
│   │               output00.txt
│   │               output01.txt
│   │               output02.txt
│   │               output03.txt
│   │               output04.txt
│   │               output05.txt
│   │               output06.txt
│   │               output07.txt
│   │               output08.txt
│   │               output09.txt
│   │               output10.txt
│   │               output11.txt
│   │               output12.txt
│   │               output13.txt
│   │               output14.txt
│   │               output15.txt
│   │               output16.txt
│   │               output17.txt
│   │               output18.txt
│   │               output19.txt
│   │               output20.txt
│   │
│   └───Bai_3_Nhap_The
│       │   Bai3.cpp
│       │   Bài toán 3 Nhập Thế - Game Kỳ Môn Thần Tốc.docx
│       │
│       └───tests
│           ├───input
│           │       input00.txt
│           │       input01.txt
│           │       input02.txt
│           │       input03.txt
│           │       input04.txt
│           │       input05.txt
│           │       input06.txt
│           │       input07.txt
│           │       input08.txt
│           │       input09.txt
│           │       input10.txt
│           │       input11.txt
│           │       input12.txt
│           │       input13.txt
│           │       input14.txt
│           │       input15.txt
│           │       input16.txt
│           │       input17.txt
│           │       input18.txt
│           │       input19.txt
│           │       input20.txt
│           │
│           └───output
│                   output00.txt
```

```
│                   output01.txt
│                   output02.txt
│                   output03.txt
│                   output04.txt
│                   output05.txt
│                   output06.txt
│                   output07.txt
│                   output08.txt
│                   output09.txt
│                   output10.txt
│                   output11.txt
│                   output12.txt
│                   output13.txt
│                   output14.txt
│                   output15.txt
│                   output16.txt
│                   output17.txt
│                   output18.txt
│                   output19.txt
│                   output20.txt
│
├───include
│   ├───Algorithms
│   │       ThienCoEngine.h
│   │
│   ├───Audio
│   │       SoundManager.h
│   │
│   ├───Core
│   │       Config.h
│   │       GameEngine.h
│   │
│   ├───Entities
│   │       GameObject.h
│   │       Map.h
│   │       Player.h
│   │
│   └───Graphics
│           TextureManager.h
│
└───src
    │   main.cpp
    │
    ├───Algorithms
    │       ThienCoEngine.cpp
    │
    ├───Audio
    │       SoundManager.cpp
    │
    ├───Core
    │       GameEngine.cpp
    │
    ├───Entities
    │       Map.cpp
    │       Player.cpp
    │
```

```
    └──Graphics
            TextureManager.cpp
```

Tất cả file code:

1. File ThienCoEngine.h

```
#pragma once

// THƯ VIỆN CHUẨN
#include <vector>
#include <queue>
#include <algorithm>
#include <iostream>

// Include Map để AI có thể truy cập dữ liệu địa hình
#include "Entities/Map.h"

// CẤU TRÚC DỮ LIỆU

// Ma trận khoảng cách giữa các điểm quan trọng (Vị trí xuất phát + Các trận
nhãn)
struct Matrix {
    // Ma trận vuông [N][N] lưu khoảng cách
    // N là số lượng trận nhãn. Index 0 là vị trí xuất phát
    // distance[i][j] = Số bước đi từ điểm i đến điểm j
    std::vector<std::vector<int>> distances;

    // Tổng số lượng điểm quan trọng (1 điểm xuất phát + N trận nhãn)
    int nodeCount;

    // Hàm khởi tạo
    Matrix() : nodeCount(0) {}
};

// Kết quả tối ưu của thuật toán Thiên Mệnh (Bài toán TSP)
struct Solution {
    // Tổng số bước đi nhỏ nhất để hoàn thành màn chơi (Kỷ lục Rank S)
    int totalSteps;

    // Thứ tự các điểm cần đi qua để đạt được số bước trên
    // Ví dụ: {0, 2, 1, 3} nghĩa là: Xuất phát -> trận nhãn 2 -> trận nhãn 1
-> trận nhãn 3
    std::vector<int> pathSequence;

    Solution() : totalSteps(1e9) {}
};

// CLASS: ThienCoEngine (MẪU THIẾT KẾ SINGLETON)
class ThienCoEngine {
public:
    // Đảm bảo chỉ có 1 bộ não AI duy nhất trong suốt vòng đời game
    static ThienCoEngine* GetInstance();
    static void DestroyInstance();

    // TÍNH TOÁN MA TRẬN KHOẢNG CÁCH GIỮA CÁC ĐIỂM QUAN TRỌNG (Vị trí xuất
phát + Các trận nhãn)
```

```
    Matrix CalculateMatrix(Map* pMap);

    // TÍNH TOÁN KẾT QUẢ TỐI ƯU THEO THIÊN MỆNH (BÀI TOÁN TSP)
    Solution CalculateSolution(const Matrix& matrix);

private:
    // Hàm khởi tạo riêng tư
    ThienCoEngine();
    ~ThienCoEngine();

    static ThienCoEngine* s_Instance;

    // Mảng 2 chiều lưu khoảng cách và đánh dấu đã thăm
    // Giá trị -1 nghĩa là chưa thăm
    std::vector<std::vector<int>> dist;

    // THUẬT TOÁN BFS (TÌM ĐƯỜNG NGẮN NHẤT TRÊN LƯỚI)
    int BFS_FindShortestPath(MapPoint start, MapPoint end, Map* pMap);

    // THUẬT TOÁN QUY HOẠCH ĐỘNG VỚI BITMASK GIẢI BÀI TOÁN TSP
    void SolveTSP_DP(const Matrix& matrix, Solution& solution);

    // TRUY VẾT ĐƯỜNG ĐI TỪ KẾT QUẢ QUY HOẠCH ĐỘNG
    std::vector<int> TracebackPath(const std::vector<std::vector<int>>& dp,
                                   const std::vector<std::vector<int>>& parent,
                                   int nodeCount, int finalMask, int lastNode);

    // HẰNG SỐ
    const int START_NODE_INDEX = 0; // Node 0 luôn được quy định là vị trí
bắt đầu của nhân vật
    const int INF = 1e9;
};
```

2. File ThienCoEngine.cpp

```
#include "Algorithms/ThienCoEngine.h"

// Khởi tạo con trỏ Singleton
ThienCoEngine* ThienCoEngine::s_Instance = nullptr;

// TRIỂN KHAI MẪU THIẾT KẾ SINGLETON (SINGLETON PATTERN IMPLEMENTATION)

ThienCoEngine* ThienCoEngine::GetInstance() {
    if (s_Instance == nullptr) s_Instance = new ThienCoEngine();
    return s_Instance;
}

void ThienCoEngine::DestroyInstance() {
    if (s_Instance != nullptr) {
        delete s_Instance;
        s_Instance = nullptr;
    }
}

// Hàm khởi tạo riêng tư (Private Constructor)
ThienCoEngine::ThienCoEngine() {}
```

```
ThienCoEngine::~ThienCoEngine() {}

// TÍNH TOÁN MA TRẬN KHOẢNG CÁCH GIỮA CÁC ĐIỂM QUAN TRỌNG (Vị trí xuất phát
+ Các trận nhãn)
Matrix ThienCoEngine::CalculateMatrix(Map* pMap) {
    Matrix matrix;

    // 1. Danh sách các điểm quan trọng (Nodes)
    // Node 0: Vị trí xuất phát của nhân vật (Start Point)
    // Node 1..N: Các trận nhãn (Shrines)
    std::vector<MapPoint> keyPoints;
     keyPoints.push_back(pMap->GetStartPoint()); // Thêm điểm xuất phát vào
đầu tiên

    const std::vector<MapPoint>& shrines = pMap->GetShrines();
    keyPoints.insert(keyPoints.end(), shrines.begin(), shrines.end());

    int N = (int)keyPoints.size();
    matrix.nodeCount = N;

    // 2. Khởi tạo kích thước ma trận khoảng cách (NxN)
    // Giá trị ban đầu là 0
    matrix.distances.assign(N, std::vector<int>(N, 0));

    // 3. Chạy BFS giữa mọi cặp điểm để tính khoảng cách ngắn nhất
     // Vì đồ thị vô hướng (đi từ A->B bằng B->A), chỉ cần tính một nửa ma
trận
    for (int i = 0; i < N; i++) {
        for (int j = i + 1; j < N; j++) {
            // Tính khoảng cách từ Node i đến Node j
           int dist = BFS_FindShortestPath(keyPoints[i], keyPoints[j], pMap);

            // Lưu vào ma trận đối xứng (vì đồ thị vô hướng)
            matrix.distances[i][j] = dist;
            matrix.distances[j][i] = dist;
        }
    }
    return matrix;
}

// THUẬT TOÁN BFS (TÌM ĐƯỜNG NGẮN NHẤT TRÊN LƯỚI)
int ThienCoEngine::BFS_FindShortestPath(MapPoint start, MapPoint end, Map*
pMap) {
    if (start.row == end.row && start.col == end.col) return 0;

    int rows = pMap->GetRows();
    int cols = pMap->GetCols();

    // Thay đổi kích thước hoặc đặt lại ma trận khoảng cách
     if (dist.size() != (size_t)rows || (rows > 0 && dist[0].size() !=
(size_t)cols)) {
        dist.assign(rows, std::vector<int>(cols, -1));
    } else {
        for (int i = 0; i < rows; i++) {
            std::fill(dist[i].begin(), dist[i].end(), -1);
```

```
        }
    }

    std::queue<MapPoint> q;
    q.push(start);
    dist[start.row][start.col] = 0;

    // 4 hướng: Lên, Xuống, Trái, Phải
    int dRow[] = {-1, 1, 0, 0};
    int dCol[] = {0, 0, -1, 1};

    while (!q.empty()) {
        MapPoint current = q.front();
        q.pop();

        if (current.row == end.row && current.col == end.col) {
            return dist[current.row][current.col];
        }

        for (int i = 0; i < 4; i++) {
            int newRow = current.row + dRow[i];
            int newCol = current.col + dCol[i];

            // Kiểm tra biên và tính hợp lệ
            if (newRow >= 0 && newRow < rows && newCol >= 0 && newCol < cols)
{
                    if (dist[newRow][newCol] == -1 && pMap->GetTileID(newRow,
newCol) != 1) {
                       dist[newRow][newCol] = dist[current.row][current.col] +
1;
                      q.push({newRow, newCol});
                }
            }
        }
    }

    return INF;
}

// TÍNH TOÁN KẾT QUẢ TỐI ƯU THEO THIÊN MỆNH (BÀI TOÁN TSP)
Solution ThienCoEngine::CalculateSolution(const Matrix& matrix) {
    Solution solution;
    int N = matrix.nodeCount;

    // Trường hợp biên: Nếu chỉ có điểm xuất phát (N = 1) hoặc không có điểm
nào
    if (N <= 1) {
        solution.totalSteps = 0;
        return solution;
    }

    // Gọi thuật toán Quy hoạch động với Bitmask
    SolveTSP_DP(matrix, solution);
    return solution;
}
```

```
// THUẬT TOÁN DP BITMASK GIẢI BÀI TOÁN TSP
void ThienCoEngine::SolveTSP_DP(const Matrix& matrix, Solution& solution) {
    int N = matrix.nodeCount;
    int limitMask = 1 << N; // Tổng số trạng thái: 2^N

    // 1. Khởi tạo bảng DP và bảng truy vết Parent
    // dp[mask][u]: Chi phí nhỏ nhất để đi qua tập 'mask' và kết thúc tại 'u'
    std::vector<std::vector<int>> dp(limitMask, std::vector<int>(N, INF));

     // parent[mask][u]: Lưu node trước đó đã đi để đến được 'u' với trạng
thái 'mask'
    std::vector<std::vector<int>> parent(limitMask, std::vector<int>(N, -1));

    // 2. Trạng thái cơ sở (Base Case)
    // Bắt đầu tại node 0 (Start Point), mask chỉ có bit 0 bật
    dp[1][0] = 0;

    // 3. Quy hoạch động với Bitmask
    // Duyệt qua tất cả các trạng thái (mask)
    for (int mask = 1; mask < limitMask; ++mask) {

        // Duyệt qua node cuối cùng 'u' trong tập mask hiện tại
        for (int u = 0; u < N; ++u) {

            // Nếu u có trong mask (bit thứ u bật)
            if ((mask & (1 << u))) {

                // Nếu trạng thái này chưa tới được, bỏ qua
                if (dp[mask][u] == INF) continue;

                // Thử đi tiếp đến node 'v' kế tiếp
                for (int v = 0; v < N; ++v) {

                    // Nếu v chưa có trong mask (chưa thăm)
                    if (!(mask & (1 << v))) {

                        // Tính mask mới khi thêm v
                        int nextMask = mask | (1 << v);

                        // Kiểm tra khoảng cách u->v có hợp lệ không
                        int distUV = matrix.distances[u][v];
                        if (distUV != INF && distUV != -1) {

                            // Tối ưu hóa (Relaxation):
                             // Nếu đường mới đi qua v ngắn hơn đường cũ đã
biết
                            if (dp[nextMask][v] > dp[mask][u] + distUV) {
                                dp[nextMask][v] = dp[mask][u] + distUV;
                                 parent[nextMask][v] = u; // Lưu vết: Để đến
v ở trạng thái nextMask, ta đi từ u
                            }
                        }
                    }
                }
            }
        }
```

```
    }

    // 4. Tìm kết quả tốt nhất
    // Trạng thái đích: Mask full bit 1 (tất cả các bit đều bật) -> Đã thăm
hết N node
    int fullMask = limitMask - 1;
    int minTotalSteps = INF;
    int bestLastNode = -1;

    // Điểm kết thúc có thể là bất kỳ Shrine nào
    // Duyệt tất cả các node 'i' để xem kết thúc ở đâu là tối ưu nhất
    for (int i = 1; i < N; ++i) { // i chạy từ 1 vì node 0 là Start
        if (dp[fullMask][i] < minTotalSteps) {
            minTotalSteps = dp[fullMask][i];
            bestLastNode = i;
        }
    }

    // 5. Lưu kết quả vào Solution
    solution.totalSteps = minTotalSteps;
    std::cout << "[AI] Số bước tối ưu: " << minTotalSteps << std::endl;
    // Truy vết lại đường đi từ kết quả DP
       solution.pathSequence   =   TracebackPath(dp,   parent,   N,   fullMask,
bestLastNode);
}

// TRUY VẾT ĐƯỜNG ĐI (TRACEBACK PATH)
std::vector<int> ThienCoEngine::TracebackPath(const
std::vector<std::vector<int>>& dp, const std::vector<std::vector<int>>&
parent, int nodeCount, int finalMask, int lastNode) {
    std::vector<int> path;

    // Trường hợp không tìm thấy đường đi
    if (lastNode == -1 || dp[finalMask][lastNode] >= INF) {
        std::cout << "[AI] Không tìm thấy giải pháp!" << std::endl;
        return path; // Trả về rỗng
    }

    int currMask = finalMask;
    int currNode = lastNode;

    // Lần ngược từ trạng thái cuối về trạng thái đầu
    while (currNode != -1) {
        path.push_back(currNode);

        int prevNode = parent[currMask][currNode];

        // Loại bỏ bit của currNode ra khỏi mask để quay về trạng thái trước
        int prevMask = currMask ^ (1 << currNode);

        currNode = prevNode;
        currMask = prevMask;

        // Nếu quay về mask = 0 thì dừng
        if (currMask == 0) break;
    }
```

```
    // Thêm điểm xuất phát (Node 0) vào cuối danh sách
    if (path.back() != 0) path.push_back(0);
    // Đảo ngược vector để có thứ tự đúng
    std::reverse(path.begin(), path.end());

    // In lộ trình để kiểm tra (debug)
    std::cout << "Chuỗi đường đi tối ưu: ";
    for (int i = 0; i < (int)path.size() - 1; i++) {
        std::cout << path[i] << " -> ";
    }
    std::cout << path.back() << std::endl;

    return path;
}
```

3. File SoundManager.h

```
#pragma once

// THƯ VIỆN CẦN THIẾT
#include <SDL_mixer.h>
#include <string>
#include <map>
#include <iostream>
#include <vector>

// CLASS: SoundManager (MẪU THIẾT KẾ SINGLETON)
class SoundManager {
public:
    // Lấy thể hiện duy nhất của SoundManager
    static SoundManager* GetInstance();

    // Dọn dẹp hệ thống âm thanh
    void Clean();

    // Tải nhạc nền (Music - file dài, định dạng mp3)
    bool LoadMusic(const std::string& fileName, const std::string& id);

    // Tải hiệu ứng âm thanh (SFX - file ngắn, định dạng wav)
    bool LoadSFX(const std::string& fileName, const std::string& id);

    // Phát nhạc nền
    void PlayMusic(const std::string& id); // Mặc định lặp vô tận

    // Phát hiệu ứng âm thanh
    void PlaySFX(const std::string& id);   // Mặc định phát 1 lần

    // Bật / Tắt toàn bộ âm thanh
    void SetMute(bool isMuted);

private:
    SoundManager();
    ~SoundManager();

    // Con trỏ thể hiện duy nhất (Singleton instance)
```

```
    static SoundManager* s_Instance;

     // Cấu trúc Map lưu trữ nhạc nền: Khóa = Tên ID, Giá trị = Con trỏ
Mix_Music*
    std::map<std::string, Mix_Music*> m_musicMap;

    // Cấu trúc Map lưu trữ hiệu ứng âm thanh: Khóa = Tên ID, Giá trị = Con
trỏ Mix_Chunk*
    std::map<std::string, Mix_Chunk*> m_sfxMap;
};
```

4. File SoundManager.cpp

```
#include "Audio/SoundManager.h"
#include <iostream>

// Khởi tạo con trỏ Singleton
SoundManager* SoundManager::s_Instance = nullptr;

SoundManager* SoundManager::GetInstance() {
    if (s_Instance == nullptr) s_Instance = new SoundManager();
    return s_Instance;
}

SoundManager::SoundManager() {
    std::cout << "[ÂM THANH] Đang khởi tạo hệ thống âm thanh..." << std::endl;
}

SoundManager::~SoundManager() { Clean(); }

void SoundManager::Clean() {
    // 1. Giải phóng nhạc nền (BGM)
    for (auto const& [id, music] : m_musicMap) Mix_FreeMusic(music);
    m_musicMap.clear();

    // 2. Giải phóng hiệu ứng âm thanh (SFX)
    for (auto const& [id, chunk] : m_sfxMap) Mix_FreeChunk(chunk);
    m_sfxMap.clear();

    // 3. Đóng thiết bị âm thanh
    Mix_CloseAudio();
    std::cout << "[ÂM THANH] Hệ thống đã được dọn dẹp và tắt." << std::endl;
}

bool SoundManager::LoadMusic(const std::string& fileName, const std::string&
id) {
    // Tải file nhạc (chế độ phát trực tiếp - Stream mode)
    Mix_Music* pMusic = Mix_LoadMUS(fileName.c_str());
    if (pMusic == nullptr) {
        std::cerr << "[LỖI] Không thể tải nhạc nền: " << fileName << " | Lỗi:
" << Mix_GetError() << std::endl;
        return false;
    }

    // Lưu vào Map
    m_musicMap[id] = pMusic;
```

```
    return true;
}

bool SoundManager::LoadSFX(const std::string& fileName, const std::string&
id) {
    // Tải file hiệu ứng (tải vào RAM)
    Mix_Chunk* pChunk = Mix_LoadWAV(fileName.c_str());
    if (pChunk == nullptr) {
        std::cerr << "[LỖI] Không thể tải hiệu ứng âm thanh: " << fileName <<
" | Lỗi: " << Mix_GetError() << std::endl;
        return false;
    }

    // Lưu vào Map
    m_sfxMap[id] = pChunk;
    return true;
}

void SoundManager::PlayMusic(const std::string& id) {
    // Tìm nhạc trong Map
    if (m_musicMap.find(id) != m_musicMap.end()) {
        Mix_PlayMusic(m_musicMap[id], -1); // -1 = Lặp vô tận
    } else {
         std::cerr << "[CẢNH BÁO] Không tìm thấy ID nhạc nền: " << id <<
std::endl;
    }
}

void SoundManager::PlaySFX(const std::string& id) {
    // Tìm SFX trong Map
    if (m_sfxMap.find(id) != m_sfxMap.end()) {
        Mix_PlayChannel(-1, m_sfxMap[id], 0); // 0 = Phát 1 lần
    } else {
        std::cerr << "[CẢNH BÁO] Không tìm thấy ID hiệu ứng âm thanh: " << id
<< std::endl;
    }
}

// CHỨC NĂNG BẬT/TẮT ÂM THANH
void SoundManager::SetMute(bool isMuted) {
    if (isMuted) {
        // Tắt tiếng: Đặt âm lượng về 0
        Mix_VolumeMusic(0);
        Mix_Volume(-1, 0); // -1 là tất cả kênh SFX
        std::cout << "[ÂM THANH] Đã tắt tiếng." << std::endl;
    } else {
        // Bật tiếng: Set volume về Max (128)
        Mix_VolumeMusic(128);
        Mix_Volume(-1, 128);
        std::cout << "[ÂM THANH] Đã bật tiếng." << std::endl;
    }
}
```

5. File Config.h

```
#pragma once
```

```
/* Sử dụng constexpr (Biểu thức hằng số) thay vì #define hay const là một kỹ thuật tối ưu hóa hiện đại của C++ giúp trình biên dịch tính toán giá trị ngay tại thời điểm biên dịch (Compile time) giảm tải cho CPU khi chạy game (Runtime) */

namespace Config {

    // 1. THIẾT LẬP HIỂN THỊ & CỬA SỔ (DISPLAY & WINDOW SETTINGS)

    // Tiêu đề hiển thị trên thanh tiêu đề cửa sổ
    static const char* const WINDOW_TITLE = "Ky Mon Than Toc";

    // Chiều rộng màn hình thiết kế (Độ phân giải logic)
    // Game sẽ luôn kết xuất ở độ phân giải này, sau đó co giãn lên màn hình thực tế
    static constexpr int SCREEN_WIDTH = 1280;

    // Chiều cao màn hình thiết kế (Độ phân giải logic)
    static constexpr int SCREEN_HEIGHT = 720;

    // Chế độ toàn màn hình mặc định khi khởi động (true/false)
    static constexpr bool FULLSCREEN_ON_START = true;

    // Tỷ lệ phóng kết xuất (chỉ ảnh hưởng hiển thị, không đổi logic)
    // Ví dụ: 1.0 = nguyên gốc, 1.5 = phóng 150%, 2.0 = phóng 200%
    static constexpr float RENDER_SCALE = 2.0f;

    // 2. THIẾT LẬP HIỆU SUẤT & THỜI GIAN (PERFORMANCE & TIMING)

    // Tốc độ khung hình mục tiêu (Frames Per Second)
    static constexpr int TARGET_FPS = 60;

    // Thời gian tối đa cho phép của một khung hình (tính bằng mili giây)
    // Công thức: 1000ms / 60fps ≈ 16.66ms
    static constexpr int FRAME_DELAY = 1000 / TARGET_FPS;

    // 3. THIẾT LẬP BẢN ĐỒ & LƯỚI (MAP & GRID SETTINGS)

    // Kích thước một ô gạch (Tile Size) tính bằng pixel
    static constexpr int TILE_SIZE = 64;

    // Khoảng trống phía dưới bản đồ (pixel) để chừa chỗ cho HUD/tooltip
    static constexpr int MAP_BOTTOM_MARGIN = 36;

    // Kích thước nhân vật (Collision Box - Hộp va chạm)
    static constexpr int PLAYER_WIDTH = 96;
    static constexpr int PLAYER_HEIGHT = 80;

    // 4. THIẾT LẬP LOGIC TRÒ CHƠI (GAMEPLAY MECHANICS)

    // Tốc độ di chuyển của nhân vật (Pixel per second - Pixel mỗi giây)
    // Giá trị này nhân với DeltaTime để ra quãng đường di chuyển
    // Điều chỉnh để vẫn ~3.5 ô/giây với TILE_SIZE mới (3.5 * 80 = 280)
    static constexpr float PLAYER_SPEED = 280.0f;
```

```
    // Các định danh loại ô trên bản đồ (Tile IDs)
    // Đồng bộ với file bản đồ .txt và logic xử lý
    namespace TileID {
        static constexpr int GROUND = 0; // Đường đi / Đã ăn
        static constexpr int AIR    = 1;  // Chân không (Gây thua)
        static constexpr int SHRINE = 2; // Trận Nhãn (Mục tiêu cần thu thập)
        static constexpr int START  = 0; // Vị trí bắt đầu
    }

    // 5. ĐƯỜNG DẪN TÀI NGUYÊN (RESOURCE PATHS)

    // Đường dẫn gốc
    static const char* const ASSET_DIR = "assets/";

    // Đường dẫn chi tiết
    static const char* const LEVEL_DIR = "assets/levels/";
    static const char* const TEXTURE_DIR = "assets/images/";
    static const char* const AUDIO_DIR = "assets/audio/";
    static const char* const FONT_PATH = "assets/fonts/Roboto-Regular.ttf";
}
```

6. File GameEngine.h

```
#pragma once

// THƯ VIỆN CHUẨN & SDL
#include <SDL.h>
#include <vector>
#include <stack>
#include <string>

// CÁC MODULE HỆ THỐNG
#include "Graphics/TextureManager.h"

// Forward Declaration để tránh vòng lặp include
class Player;
class ThienCoEngine;
class Map;

// TRẠNG THÁI GAME (GAME STATES)
enum GameState {
    STATE_INTRO,            // Màn hình giới thiệu
    STATE_STORY_NAME,       // Màn hình hiển thị tên màn chơi
    STATE_STORY_SENSEI,     // Màn hình hiển thị lời dạy của thầy
    STATE_PLAY,             // Trạng thái chơi game
    STATE_RESULT_S,         // Màn hình kết quả thắng rank S
    STATE_RESULT_A,         // Màn hình kết quả thắng rank A
    STATE_RESULT_LOSE,      // Màn hình kết quả thua
    STATE_CONTINUE          // Màn hình tiếp tục sau vòng 3 (chuyển sang vòng
4+)
};

// SNAPSHOT (DÙNG CHO TÍNH NĂNG HOÀN TÁC - UNDO)
// Lưu lại trạng thái của nhân vật và bản đồ tại một thời điểm
struct GameStateMoment {
```

```
    int playerGridRow;       // Vị trí dòng của nhân vật trên lưới (đơn vị: ô lưới)
    int playerGridCol;       // Vị trí cột của nhân vật trên lưới (đơn vị: ô lưới)
    int currentSteps;        // Số bước đi hiện tại
    int shrinesCollected;    // Số trận nhãn đã thu thập
    int currentLevelIdx;     // Chỉ số level hiện tại (để khôi phục đúng level khi undo)

    // Danh sách tọa độ các trận nhãn đã được kích hoạt tại thời điểm này
    std::vector<std::pair<int, int>> visitedShrinesSnapshot;
};

// CLASS: GameEngine (MẪU THIẾT KẾ SINGLETON)
class GameEngine {
public:
    // Đảm bảo chỉ có duy nhất một thể hiện của GameEngine tồn tại trong toàn chương trình
    static GameEngine* GetInstance();
    static void DestroyInstance();

    // VÒNG ĐỜI ỨNG DỤNG (APPLICATION LIFECYCLE)

    // Khởi tạo toàn bộ hệ thống (SDL, Cửa sổ, Bộ kết xuất, Tài nguyên)
    // Trả về true nếu thành công, false nếu thất bại
    bool Init(const char* title, int xPos, int yPos, int width, int height, bool fullscreen);

    // Xử lý sự kiện đầu vào (Bàn phím, Sự kiện cửa sổ)
    void HandleEvents();

    // Cập nhật logic game (AI, Vật lý, Hoạt ảnh, Chuyển đổi trạng thái)
    void Update();

    // Vẽ toàn bộ khung hình hiện tại lên màn hình
    void Render();

    // Dọn dẹp tài nguyên trước khi thoát (Giải phóng bộ nhớ)
    void Clean();

    // Kiểm tra game còn chạy không (để duy trì vòng lặp chính)
    bool IsRunning() const { return m_bRunning; }

    // HỆ THỐNG TRUY CẬP (GETTERS)

    SDL_Renderer* GetRenderer() const { return m_pRenderer; }

    // Thời gian giữa 2 khung hình (Delta Time) - Dùng cho hoạt ảnh
    float GetDeltaTime() const { return m_deltaTime; }

    // Truy cập các đối tượng game
    Map* GetMap() const { return m_pMap; }
    Player* GetPlayer() const { return m_pPlayer; }

    // ĐIỀU KHIỂN LOGIC TRÒ CHƠI (GAMEPLAY LOGIC CONTROL)
```

```
    // Chuyển đổi trạng thái game (có lưu lịch sử để hoàn tác, có hiệu ứng chuyển cảnh)
    void SwitchState(GameState newState);

    // Hệ thống quản lý cấp độ (Level Management)
    void LoadLevel(int levelIndex); // Tải màn chơi theo chỉ số
    void NextLevel();               // Chuyển sang màn kế tiếp

    // Các tính năng trò chơi (Gameplay Features)
    void SaveState();   // Lưu trạng thái hiện tại vào ngăn xếp (Stack)
    void Undo();        // Xử lý hoàn tác tổng hợp
    void ToggleAudio(); // Bật/tắt âm thanh

    // Xử lý sự kiện gameplay
    void OnPlayerMove();                    // Gọi khi nhân vật di chuyển
    void OnShrineVisited(int row, int col); // Gọi khi nhân vật thu thập trận nhãn

    // QUẢN LÝ THÔNG SỐ TRÒ CHƠI (GAMEPLAY STATISTICS)
    int GetCurrentSteps() const { return m_currentSteps; }
    int GetOptimalSteps() const { return m_optimalSteps; }
    int GetShrinesCollected() const { return m_shrinesCollected; }
    int GetTotalShrines() const { return m_totalShrines; }

private:
    // Hàm khởi tạo riêng tư (Mẫu thiết kế Singleton)
    GameEngine();
    ~GameEngine();

    // Biến tĩnh lưu trữ thể hiện duy nhất
    static GameEngine* s_Instance;

    // CÁC THÀNH PHẦN CỐT LÕI (CORE COMPONENTS)
    bool m_bRunning;                // Cờ kiểm soát vòng lặp game
    SDL_Window* m_pWindow;          // Cửa sổ ứng dụng
    SDL_Renderer* m_pRenderer;      // Bộ kết xuất đồ họa

    int m_windowWidth;              // Chiều rộng cửa sổ
    int m_windowHeight;             // Chiều cao cửa sổ

    float m_deltaTime;                // Thời gian trôi qua giữa khung hình trước và khung hình hiện tại (giây)
    Uint64 m_lastTime;              // Mốc thời gian của khung hình trước
    Uint64 m_performanceFrequency;  // Tần số bộ đếm hiệu suất CPU

    // CÁC ĐỐI TƯỢNG GAME (GAME OBJECTS)
    Player* m_pPlayer;              // Nhân vật chính
    Map* m_pMap;                    // Bản đồ (Tilemap)

    // QUẢN LÝ CẤP ĐỘ (LEVEL MANAGEMENT)
    int m_currentLevelIdx;                  // Chỉ số cấp độ hiện tại (bắt đầu từ 1, 2, 3...)
    std::vector<std::string> m_levelFiles;  // Danh sách đường dẫn file bản đồ

    int m_currentSteps;             // Số bước người chơi đã đi
```

```
    int m_optimalSteps;              // Số bước tối ưu (tính bằng thuật toán
AI ThienCoEngine)
    int m_shrinesCollected;          // Số trận nhãn đã thu thập
    int m_totalShrines;              // Tổng số trận nhãn trong bản đồ
    int m_resultLevelIdx;            // Lưu level dùng để hiển thị màn hình
kết quả
    std::vector<std::pair<int, int>> m_visitedShrinesList; // Danh sách tọa
độ trận nhãn đã thu thập

    // Ngăn xếp hoàn tác cho gameplay (Lưu các bước đi)
    std::stack<GameStateMoment> m_historyStack;

    // Ngăn xếp hoàn tác cho trạng thái (Lưu các màn hình đã qua)
    std::stack<GameState> m_stateHistory;

    // Lưu trạng thái game trước khi rời khỏi STATE_PLAY
    // Dùng để quay lại game khi nhấn U từ màn hình INTRO/RESULT
    bool m_hasPreviousGameState = false;
    GameStateMoment m_previousGameState;

    // QUẢN LÝ TRẠNG THÁI (STATE MANAGEMENT)
    GameState m_currentState; // Trạng thái hiện tại
    GameState m_nextState;  // Trạng thái chờ chuyển đến

    // HIỆU ỨNG HÌNH ẢNH (VISUAL EFFECTS)

    // Hiệu ứng chuyển cảnh (Transition - mờ dần)
    float m_fadeAlpha;      // Độ mờ của lớp phủ màu đen (0.0 -> 255.0)
    bool m_isFadingIn;      // Đang sáng dần lên?
    bool m_isFadingOut;     // Đang tối dần đi?
    const float FADE_SPEED = 2.5f; // Tốc độ hiệu ứng chuyển cảnh

    // Chuỗi hiệu ứng chiến thắng (Victory Sequence)
    bool m_isWinningSequence;
    float m_winDelayTimer;

    // Trạng thái âm thanh
    bool m_isMuted = true;

    // Biến lưu ID hình ảnh ngẫu nhiên cho các vòng sau vòng 3
    std::string m_randomRankImageID;

    // Cờ chờ tăng level (dùng để tránh hiển thị sai ảnh trong quá trình fade)
    bool m_isPendingNextLevel = false;

    // CÁC HÀM TRỢ GIÚP (HELPER FUNCTIONS)
    // Các hàm vẽ nội bộ
    void DrawBackground();  // Vẽ nền
    std::string GetCurrentStoryImageID(const std::string& suffix) const; //
Hàm lấy ID hình ảnh theo cấp độ
    void HandleStoryInput(SDL_Event& event); // Xử lý phím ENTER chung cho
các màn hình cốt truyện
};
```

7. File GameEngine.cpp

```
#include "Core/GameEngine.h"
#include "Core/Config.h"
#include "Entities/Player.h"
#include "Entities/Map.h"
#include "Graphics/TextureManager.h"
#include "Audio/SoundManager.h"
#include <SDL_image.h>
#include <SDL_ttf.h>
#include <SDL_mixer.h>
#include "Algorithms/ThienCoEngine.h"   // Bộ não xử lý AI
#include <iostream>
#include <filesystem> // Thư viện quét file hệ thống
#include <algorithm>  // Thư viện sắp xếp
#include <regex>      // Để lọc lấy số từ tên file (level10 -> 10)

namespace fs = std::filesystem;

// Khởi tạo biến tĩnh (Static variable initialization)
GameEngine* GameEngine::s_Instance = nullptr;

// TRIỂN KHAI MẪU THIẾT KẾ SINGLETON (SINGLETON PATTERN IMPLEMENTATION)

GameEngine* GameEngine::GetInstance() {
    if (s_Instance == nullptr) s_Instance = new GameEngine();
    return s_Instance;
}

void GameEngine::DestroyInstance() {
    if (s_Instance != nullptr) {
        s_Instance->Clean();
        delete s_Instance;
        s_Instance = nullptr;
    }
}

// Hàm khởi tạo (Constructor): Thiết lập giá trị mặc định
GameEngine::GameEngine()
    : m_bRunning(false), m_pWindow(nullptr), m_pRenderer(nullptr),
      m_deltaTime(0.0f), m_lastTime(0), m_performanceFrequency(0),
      m_pPlayer(nullptr), m_pMap(nullptr),
    m_currentLevelIdx(1),
    m_currentSteps(0), m_optimalSteps(0),
    m_shrinesCollected(0), m_totalShrines(0),
    m_resultLevelIdx(1),
      m_currentState(STATE_INTRO),
      m_fadeAlpha(255.0f), m_isFadingIn(true), m_isFadingOut(false),
      m_isWinningSequence(false), m_winDelayTimer(0.0f),
      m_isMuted(false), m_isPendingNextLevel(false)
{
    // TỰ ĐỘNG QUÉT DANH SÁCH LEVEL
    std::string levelFolderPath = "assets/levels/";
    m_levelFiles.clear();
    try {
        if (fs::exists(levelFolderPath) && fs::is_directory(levelFolderPath))
{
```

```
                std::cout << "[HỆ THỐNG] Đang quét các cấp độ trong: " <<
levelFolderPath << std::endl;

            // 1. Quét tất cả file trong thư mục
            for (const auto& entry : fs::directory_iterator(levelFolderPath))
{
                // Chỉ lấy file thường (không lấy thư mục con) và có đuôi .txt
                  if (entry.is_regular_file() && entry.path().extension() ==
".txt") {
                    // Chuyển path về dạng string chuẩn (dùng / thay vì \)
                    std::string pathStr = entry.path().generic_string();
                    m_levelFiles.push_back(pathStr);
                }
            }

            // 2. Sắp xếp file: level1 -> level2 -> ... level10
               std::sort(m_levelFiles.begin(), m_levelFiles.end(), [](const
std::string& a, const std::string& b) {
                // Lambda tách số từ tên file: "level10.txt" -> 10
                static const std::regex re("(\\d+)");
                std::smatch matchA, matchB;

                int numA = 9999, numB = 9999;
                            if (std::regex_search(a, matchA, re)) numA =
std::stoi(matchA.str());
                            if (std::regex_search(b, matchB, re)) numB =
std::stoi(matchB.str());

                return numA < numB;
            });

            // In ra danh sách đã tải để kiểm tra
            for (const auto& file : m_levelFiles) {
                std::cout << "  -> Đã tải: " << file << std::endl;
            }
                std::cout << "[HỆ THỐNG] Tổng số cấp độ đã tải: " <<
m_levelFiles.size() << std::endl;

        } else {
                std::cerr << "[LỖI] Không tìm thấy thư mục cấp độ: " <<
levelFolderPath << std::endl;
        }
    } catch (const fs::filesystem_error& e) {
         std::cerr << "[NGOẠI LỆ] Lỗi hệ thống tập tin: " << e.what() <<
std::endl;
    }
}

GameEngine::~GameEngine() { Clean(); }

// VÒNG ĐỜI ỨNG DỤNG (APPLICATION LIFECYCLE)

bool GameEngine::Init(const char* title, int xPos, int yPos, int width, int
height, bool fullscreen) {
    // 1. Khởi tạo thư viện SDL2 (Video, Audio, Timer)
    if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO | SDL_INIT_TIMER) != 0) {
```

```
            std::cerr << "[LỖI NGHIÊM TRỌNG] Khởi tạo SDL thất bại: " <<
SDL_GetError() << std::endl;
        return false;
    }

    // 2. Cấu hình thuộc tính cửa sổ (OpenGL, Có thể thay đổi kích thước)
    Uint32 flags = SDL_WINDOW_SHOWN | SDL_WINDOW_RESIZABLE;
    if (fullscreen) flags |= SDL_WINDOW_FULLSCREEN_DESKTOP; // Chế độ toàn
màn hình không viền

    // 3. Tạo cửa sổ ứng dụng
    m_pWindow = SDL_CreateWindow(title, xPos, yPos, width, height, flags);
    if (!m_pWindow) {
            std::cerr << "[LỖI NGHIÊM TRỌNG] Tạo cửa sổ thất bại: " <<
SDL_GetError() << std::endl;
        return false;
    }

    // 4. Tạo bộ kết xuất đồ họa (Renderer)
    m_pRenderer = SDL_CreateRenderer(m_pWindow, -1, SDL_RENDERER_ACCELERATED
| SDL_RENDERER_PRESENTVSYNC);
    if (!m_pRenderer) {
        std::cerr << "[LỖI NGHIÊM TRỌNG] Tạo bộ kết xuất đồ họa thất bại: "
<< SDL_GetError() << std::endl;
        return false;
    }

    // Thiết lập độ phân giải ảo (Logical Resolution)
     // Game sẽ luôn kết xuất ở độ phân giải Config::SCREEN_WIDTH x
Config::SCREEN_HEIGHT,
    // nhưng tự động co giãn mượt mà lên màn hình 2K/4K
           SDL_RenderSetLogicalSize(m_pRenderer,     Config::SCREEN_WIDTH,
Config::SCREEN_HEIGHT);
    SDL_SetHint(SDL_HINT_RENDER_SCALE_QUALITY, "1"); // 1 = Linear (Mượt), 0
= Nearest (Pixel Art)
    // Không đặt scale toàn cục để UI, nền giữ nguyên kích thước

    // 5. Khởi tạo các thư viện mở rộng
    if (!(IMG_Init(IMG_INIT_PNG | IMG_INIT_JPG))) {
        std::cerr << "[LỖI NGHIÊM TRỌNG] Khởi tạo SDL_image thất bại: " <<
IMG_GetError() << std::endl;
        return false;
    }
    if (TTF_Init() == -1) {
        std::cerr << "[LỖI NGHIÊM TRỌNG] Khởi tạo SDL_ttf thất bại: " <<
TTF_GetError() << std::endl;
        return false;
    }
    if (Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2, 2048) < 0) {
        std::cerr << "[LỖI NGHIÊM TRỌNG] Khởi tạo SDL_mixer thất bại: " <<
Mix_GetError() << std::endl;
        return false;
    }

    m_windowWidth = width;
    m_windowHeight = height;
```

```
    // 6. Khởi tạo bộ đếm thời gian hiệu suất cao
    m_performanceFrequency = SDL_GetPerformanceFrequency();
    m_lastTime = SDL_GetPerformanceCounter();

    // 7. Khởi tạo các phân hệ con (Subsystems)
    SoundManager::GetInstance();                          // Khởi tạo quản lý
âm thanh

    // 8. Tải tài nguyên dùng chung
    // Phông chữ
        if   (!TextureManager::GetInstance()->LoadFont("assets/fonts/Roboto-
Regular.ttf", "gui_font", 32)) {
        std::cerr << "[CẢNH BÁO] Tải phông chữ thất bại!" << std::endl;
    }

    // Hàm Lambda tiện ích: Tự động tải tất cả ảnh .png trong một thư mục
    auto AutoLoadTextures = [&](const std::string& folderPath) {
        if (!fs::exists(folderPath)) {
             std::cerr << "[CẢNH BÁO] Không tìm thấy thư mục texture: " <<
folderPath << std::endl;
            return;
        }

         std::cout << "[TÀI NGUYÊN] Đang quét thư mục: " << folderPath <<
" ..." << std::endl;
        for (const auto& entry : fs::directory_iterator(folderPath)) {
            // Chỉ xử lý file .png
           if (entry.is_regular_file() && entry.path().extension() == ".png")
{
                               //  1.  Lấy  đường  dẫn  file  chuẩn  (ví  dụ:
assets/images/characters/idle_down.png)
                std::string path = entry.path().generic_string();

                // 2. Lấy tên file làm ID (ví dụ: idle_down)
                std::string id = entry.path().stem().string();

                // 3. Tải ảnh
                bool success = TextureManager::GetInstance()->Load(path, id,
m_pRenderer);

                if (success) {
                    std::cout << "  + Đã tải: " << id << std::endl;
                }
            }
        }
    };

    // Ảnh nền & UI (Quét tự động)
    AutoLoadTextures("assets/images/characters");
    AutoLoadTextures("assets/images/environment");

    // Âm thanh
     SoundManager::GetInstance()->LoadMusic("assets/audio/bgm/bgm_menu.mp3",
"bgm_menu");
```

```
     SoundManager::GetInstance()->LoadMusic("assets/audio/bgm/bgm_game.mp3",
"bgm_game");
      SoundManager::GetInstance()->LoadSFX("assets/audio/sfx/sfx_click.wav",
"sfx_click");
    SoundManager::GetInstance()->LoadSFX("assets/audio/sfx/sfx_collect.wav",
"sfx_collect");
       SoundManager::GetInstance()->LoadSFX("assets/audio/sfx/sfx_lose.wav",
"sfx_lose");
       SoundManager::GetInstance()->LoadSFX("assets/audio/sfx/sfx_step.wav",
"sfx_step");
        SoundManager::GetInstance()->LoadSFX("assets/audio/sfx/sfx_win.wav",
"sfx_win");

    // Khởi tạo Map và Player
    m_pMap = new Map();
     m_pPlayer = new Player(new LoaderParams(0, 0, Config::PLAYER_WIDTH,
Config::PLAYER_HEIGHT, "idle_down", 8, 100));

    // Bắt đầu nhạc nền Menu
    Mix_HaltMusic(); // Dừng nhạc hiện tại
    SoundManager::GetInstance()->PlayMusic("bgm_menu");

    m_bRunning = true;
    std::cout << "[HỆ THỐNG] Khởi tạo Game Engine thành công." << std::endl;
    return true;
}

void GameEngine::LoadLevel(int levelIndex) {
    int vectorIdx = levelIndex - 1;
    if (vectorIdx < 0 || vectorIdx >= (int)m_levelFiles.size()) return;

    // Đặt lại logic GAME (Reset game logic)
    while (!m_historyStack.empty()) m_historyStack.pop();
    m_visitedShrinesList.clear();
    m_currentSteps = 0;
    m_shrinesCollected = 0;
    m_currentLevelIdx = levelIndex;

    // 1. Tải bản đồ từ file
    if (!m_pMap || !m_pPlayer) return;

    m_pMap->LoadMap(m_levelFiles[vectorIdx]);

    // 2. Thiết lập vị trí nhân vật (áp dụng offset để căn giữa bản đồ)
    MapPoint startPos = m_pMap->GetStartPoint();
    // Chuyển tọa độ lưới (Grid) -> Tọa độ pixel (nhân với kích thước ô lưới)
+ offset
    m_pPlayer->SetPosition(
        startPos.col * Config::TILE_SIZE + m_pMap->GetOffsetX(),
        startPos.row * Config::TILE_SIZE + m_pMap->GetOffsetY()
    );

    // 3. Tính toán đường đi tối ưu bằng AI
    // Sử dụng ThienCoEngine để giải bài toán người bán hàng (TSP)
    std::cout << "[AI] Đang tính toán cho Cấp độ " << levelIndex + 1 << "..."
<< std::endl;
```

```
    auto matrix = ThienCoEngine::GetInstance()->CalculateMatrix(m_pMap);
    auto solution = ThienCoEngine::GetInstance()->CalculateSolution(matrix);

    m_optimalSteps = solution.totalSteps;
    m_totalShrines = (int)m_pMap->GetShrines().size();

    std::cout << "[AI] Số bước tối ưu: " << m_optimalSteps << std::endl;

    // Lưu trạng thái đầu tiên vào lịch sử hoàn tác
    SaveState();
}

std::string GameEngine::GetCurrentStoryImageID(const std::string& suffix)
const {
    // suffix ví dụ: "_name" (tên màn chơi), "_sensei" (lời dạy), "_rankS",
"_rankA"

    // Logic: index 1 -> img_scence1
    // index 2 -> img_scence2
    int sceneNum = m_currentLevelIdx;

    // Với màn hình kết quả, luôn hiển thị theo level vừa hoàn thành
    if (m_currentState == STATE_RESULT_S || m_currentState == STATE_RESULT_A
|| m_currentState == STATE_RESULT_LOSE) {
        sceneNum = m_resultLevelIdx;
    }

    if (sceneNum <= 3) {
        // Vòng 1, 2, 3: Trả về đúng hình ảnh cốt truyện
        // Ví dụ: "img_scence1_name"
        return "img_scence" + std::to_string(sceneNum) + suffix;
    } else {
        // Vòng 4 trở đi: Dùng hình ảnh ngẫu nhiên từ 3 vòng đầu cho tất cả
loại màn hình
        if (m_randomRankImageID.empty()) {
            int randomScene = 1 + (rand() % 3); // Ngẫu nhiên từ 1 đến 3
           const_cast<GameEngine*>(this)->m_randomRankImageID = "img_scence"
+ std::to_string(randomScene) + suffix;
        }
        return m_randomRankImageID;
    }
}

void GameEngine::HandleEvents() {
    SDL_Event event;
    while (SDL_PollEvent(&event)) {
        // Sự kiện hệ thống (Quit)
        if (event.type == SDL_QUIT) m_bRunning = false;

        // Phím tắt toàn cục
        if (event.type == SDL_KEYDOWN) {

            // 1. Phím '1': Về Intro và Reset về Level 1
            if (event.key.keysym.sym == SDLK_1) {
                m_currentLevelIdx = 1; // Reset về vòng 1
```

```
                SwitchState(STATE_INTRO);
            }

            // 2. Phím '2': Bật/Tắt nhạc
            if (event.key.keysym.sym == SDLK_2) ToggleAudio();

            // 3. Phím 'u': Undo
            if (event.key.keysym.sym == SDLK_u) Undo();

            // 4. Phím 'q': Quit Game
            if (event.key.keysym.sym == SDLK_q) m_bRunning = false;

            // 5. Phím 'F11': Fullscreen
            if (event.key.keysym.sym == SDLK_F11) {
                Uint32 flags = SDL_GetWindowFlags(m_pWindow);
                                if   (flags   &   SDL_WINDOW_FULLSCREEN_DESKTOP)
SDL_SetWindowFullscreen(m_pWindow, 0);
                                    else    SDL_SetWindowFullscreen(m_pWindow,
SDL_WINDOW_FULLSCREEN_DESKTOP);
            }
        }

        // Xử lý Input theo State
        switch (m_currentState) {
            case STATE_INTRO:
            case STATE_STORY_NAME:
            case STATE_STORY_SENSEI:
            case STATE_RESULT_S:
            case STATE_RESULT_A:
            case STATE_RESULT_LOSE:
            case STATE_CONTINUE:
                HandleStoryInput(event);
                break;

            default: break;
        }
    }
}

void GameEngine::Update() {
    // 1. Lấy thời gian hiện tại
    Uint64 currentTime = SDL_GetPerformanceCounter();

    // 2. Tính chênh lệch
    Uint64 diff = currentTime - m_lastTime;

    // 3. Chuyển sang giây
    m_deltaTime = (float)diff / (float)m_performanceFrequency;

    // 4. Cập nhật mốc thời gian
    m_lastTime = currentTime;

    // 5. Lag Protection: Kẹp tối đa 0.05s (20FPS) để tránh lỗi vật lý
    if (m_deltaTime > 0.05f) m_deltaTime = 0.05f;

    // 7. Hiệu ứng chuyển cảnh Fade In/Out
```

```
    if (m_isFadingOut) {
        m_fadeAlpha += FADE_SPEED * m_deltaTime * 255.0f;
        if (m_fadeAlpha >= 255.0f) {
            m_fadeAlpha = 255.0f;
            m_isFadingOut = false;
            m_isFadingIn = true;            // Bắt đầu sáng lên
            m_currentState = m_nextState;   // Chuyển trạng thái

            // Nếu vào game, dừng nhạc menu và phát nhạc game
            if (m_currentState == STATE_PLAY) {
                Mix_HaltMusic(); // Dừng nhạc hiện tại
                SoundManager::GetInstance()->PlayMusic("bgm_game");
            } else if (m_currentState == STATE_INTRO) {
                // Quay về Intro, dừng nhạc game và phát nhạc menu
                Mix_HaltMusic(); // Dừng nhạc hiện tại
                SoundManager::GetInstance()->PlayMusic("bgm_menu");
            } else if (m_currentState == STATE_RESULT_S || m_currentState ==
STATE_RESULT_A || m_currentState == STATE_RESULT_LOSE || m_currentState ==
STATE_CONTINUE) {
                // Vào màn hình kết quả hoặc continue, dừng nhạc game và phát
nhạc nền menu
                Mix_HaltMusic();
                SoundManager::GetInstance()->PlayMusic("bgm_menu");
             } else if (m_currentState == STATE_STORY_NAME || m_currentState
== STATE_CONTINUE) {
                // Reset ảnh ngẫu nhiên khi vào chu kỳ mới (name/continue ->
sensei/game -> result)
                m_randomRankImageID = "";
            }
        }
    } else if (m_isFadingIn) {
        m_fadeAlpha -= FADE_SPEED * m_deltaTime * 255.0f;
        if (m_fadeAlpha <= 0.0f) {
            m_fadeAlpha = 0.0f;
            m_isFadingIn = false;

            // Nếu vừa fade xong vào STATE_STORY_NAME và đang chờ tăng level
             if (m_currentState == STATE_STORY_NAME && m_isPendingNextLevel)
{
                m_currentLevelIdx++;
                m_isPendingNextLevel = false;
            }
        }
    }

    // 8. Logic theo từng trạng thái
    switch (m_currentState) {
        case STATE_PLAY:
            if (m_pPlayer) {
                 m_pPlayer->Update(); // Nhân vật tự xử lý di chuyển và va
chạm

                // Logic thua: Đi vào ô núi # (ID = 1)
                // Trừ offset để lấy tọa độ lưới thực tế
                 int pRow = (int)((m_pPlayer->GetY() - m_pMap->GetOffsetY())
/ Config::TILE_SIZE);
```

```
                int pCol = (int)((m_pPlayer->GetX() - m_pMap->GetOffsetX())
/ Config::TILE_SIZE);

                        if (m_pMap && m_pMap->GetTileID(pRow, pCol) ==
Config::TileID::AIR) {
                    // Âm thanh thua sẽ được phát ở Player::Falling()
                    SwitchState(STATE_RESULT_LOSE);
                }

                // Logic thắng: Thu thập hết Trận Nhãn

                // Logic sau khi thắng -> Chuyển sang màn kết quả
                if (m_isWinningSequence) {
                    m_winDelayTimer += m_deltaTime;
                    if (m_winDelayTimer > 1.0f) {
                        m_isWinningSequence = false;

                        // So sánh số bước đi với giải pháp AI
                        m_resultLevelIdx = m_currentLevelIdx;
                        if (m_currentSteps <= m_optimalSteps) {
                            SwitchState(STATE_RESULT_S); // Hạng S - Hoàn hảo
                        } else {
                             SwitchState(STATE_RESULT_A); // Hạng A - Thắng
nhưng chưa tối ưu
                        }
                    }
                }
            }
            break;

        default: break;
    }
}

void GameEngine::Render() {
    // 1. Xóa màn hình
    SDL_SetRenderDrawColor(m_pRenderer, 0, 0, 0, 255);
    SDL_RenderClear(m_pRenderer);

    std::string imgID;

    // 2. Vẽ theo trạng thái
    switch (m_currentState) {
        case STATE_INTRO:
                    TextureManager::GetInstance()->Draw("img_intro", 0, 0,
Config::SCREEN_WIDTH, Config::SCREEN_HEIGHT, m_pRenderer);
            break;

        case STATE_STORY_NAME:
            imgID = GetCurrentStoryImageID("_name");
           if (!imgID.empty()) TextureManager::GetInstance()->Draw(imgID, 0,
0, Config::SCREEN_WIDTH, Config::SCREEN_HEIGHT, m_pRenderer);
            break;

        case STATE_STORY_SENSEI:
            imgID = GetCurrentStoryImageID("_sensei");
```

```
            if (!imgID.empty()) TextureManager::GetInstance()->Draw(imgID, 0,
0, Config::SCREEN_WIDTH, Config::SCREEN_HEIGHT, m_pRenderer);
            break;

        case STATE_PLAY:
            // Vẽ Game
            DrawBackground();   // Vẽ nền
            if (m_pMap) m_pMap->DrawMap();  // Vẽ Map
            if (m_pPlayer) m_pPlayer->Draw();  // Vẽ Player
            break;

        case STATE_RESULT_S:
            imgID = GetCurrentStoryImageID("_rankS");
                        TextureManager::GetInstance()->Draw(imgID,   0,   0,
Config::SCREEN_WIDTH, Config::SCREEN_HEIGHT, m_pRenderer);
            break;

        case STATE_RESULT_A:
            imgID = GetCurrentStoryImageID("_rankA");
                        TextureManager::GetInstance()->Draw(imgID,   0,   0,
Config::SCREEN_WIDTH, Config::SCREEN_HEIGHT, m_pRenderer);
            break;

        case STATE_RESULT_LOSE:
                    TextureManager::GetInstance()->Draw("img_lose",   0,   0,
Config::SCREEN_WIDTH, Config::SCREEN_HEIGHT, m_pRenderer);
            break;

        case STATE_CONTINUE:
                  TextureManager::GetInstance()->Draw("img_continue",  0,  0,
Config::SCREEN_WIDTH, Config::SCREEN_HEIGHT, m_pRenderer);
            break;
    }

    // 3. Vẽ hiệu ứng chuyển cảnh (Fade Overlay)
    if (m_fadeAlpha > 0.0f) {
        SDL_SetRenderDrawBlendMode(m_pRenderer, SDL_BLENDMODE_BLEND);
        SDL_SetRenderDrawColor(m_pRenderer, 0, 0, 0, (Uint8)m_fadeAlpha);
        SDL_RenderFillRect(m_pRenderer, NULL);
        SDL_SetRenderDrawBlendMode(m_pRenderer, SDL_BLENDMODE_NONE);
    }

    // Đẩy ra màn hình
    SDL_RenderPresent(m_pRenderer);
}

// HELPERS

void GameEngine::DrawBackground() {
            TextureManager::GetInstance()->Draw("bg_parallax",     0,     0,
Config::SCREEN_WIDTH, Config::SCREEN_HEIGHT, m_pRenderer);
}

// XỬ LÝ SỰ KIỆN LOGIC

void GameEngine::OnPlayerMove() {
```

```
    m_currentSteps++;
    SoundManager::GetInstance()->PlaySFX("sfx_step");
}

void GameEngine::OnShrineVisited(int row, int col) {
    if (m_pMap->GetTileID(row, col) == Config::TileID::SHRINE) {
        m_shrinesCollected++;
        m_pMap->SetTileID(row, col, Config::TileID::GROUND);
        m_visitedShrinesList.push_back({row, col});

        // Kiểm tra nếu thu thập hết shrine
        if (m_shrinesCollected >= m_totalShrines) {
            // Kích hoạt trạng thái thắng ngay tại đây để tránh phát 2 lần
            if (!m_isWinningSequence) {
                m_isWinningSequence = true;
                m_winDelayTimer = 0.0f;
                SoundManager::GetInstance()->PlaySFX("sfx_win");
            }
        } else {
            // Chỉ phát sfx_collect nếu chưa thu thập hết
            SoundManager::GetInstance()->PlaySFX("sfx_collect");
        }

        std::cout << "[GAME] Đã thu thập trấn nhãn tại (" << row << "," <<
col << ")" << std::endl;
    }
}

// HỆ THỐNG UNDO

void GameEngine::SaveState() {
    GameStateMoment moment;
    // Chuyển pixel -> grid
    moment.playerGridRow = (int)((m_pPlayer->GetY() - m_pMap->GetOffsetY())
/ Config::TILE_SIZE);
    moment.playerGridCol = (int)((m_pPlayer->GetX() - m_pMap->GetOffsetX())
/ Config::TILE_SIZE);
    moment.currentSteps = m_currentSteps;
    moment.shrinesCollected = m_shrinesCollected;
    moment.currentLevelIdx = m_currentLevelIdx;
    moment.visitedShrinesSnapshot = m_visitedShrinesList;
    m_historyStack.push(moment);
}

void GameEngine::Undo() {
    // 1. Nếu đang chơi: Undo bước đi
    if (m_currentState == STATE_PLAY) {
        if (m_historyStack.size() > 1) { // Phải còn ít nhất trạng thái Start
            m_historyStack.pop(); // Bỏ trạng thái hiện tại
            GameStateMoment prev = m_historyStack.top(); // Lấy trạng thái cũ

            // Khôi phục Player
            m_pPlayer->SetPosition(prev.playerGridCol * Config::TILE_SIZE +
m_pMap->GetOffsetX(),
                                   prev.playerGridRow * Config::TILE_SIZE +
m_pMap->GetOffsetY());
```

```
            m_currentSteps = prev.currentSteps;
            m_currentLevelIdx = prev.currentLevelIdx; // Khôi phục level index

            // Khôi phục Map (Các Trận Nhãn đã ăn phải hiện lại nếu undo)
            // Cơ chế: Khôi phục lại tất cả về chưa thu thập (ID 2), sau đó
duyệt snapshot để đánh dấu đã thu thập (ID 0)
            const std::vector<MapPoint>& allShrines = m_pMap->GetShrines();
            // Khôi phục trạng thái chưa thu thập
            for (const auto& s : allShrines) m_pMap->SetTileID(s.row, s.col,
Config::TileID::SHRINE);
            // Đánh dấu lại các trận nhãn đã thu thập trong quá khứ
                for (const auto& s : prev.visitedShrinesSnapshot) m_pMap-
>SetTileID(s.first, s.second, Config::TileID::GROUND);

            m_shrinesCollected = prev.shrinesCollected;
            m_visitedShrinesList = prev.visitedShrinesSnapshot;

            std::cout << "[GAME] Hoàn tác thành công!" << std::endl;
        }
    }
    // 2. Nếu đang ở màn hình khác nhưng có trạng thái game đã lưu: Quay lại
game
    else if (m_hasPreviousGameState) {
        // Khôi phục trạng thái game
                m_pPlayer->SetPosition(m_previousGameState.playerGridCol   *
Config::TILE_SIZE + m_pMap->GetOffsetX(),
                                       m_previousGameState.playerGridRow  *
Config::TILE_SIZE + m_pMap->GetOffsetY());
        m_currentSteps = m_previousGameState.currentSteps;
        m_currentLevelIdx = m_previousGameState.currentLevelIdx; // Khôi phục
level index

        // Khôi phục Map
        const std::vector<MapPoint>& allShrines = m_pMap->GetShrines();
         for (const auto& s : allShrines) m_pMap->SetTileID(s.row, s.col,
Config::TileID::SHRINE);
        for (const auto& s : m_previousGameState.visitedShrinesSnapshot)
            m_pMap->SetTileID(s.first, s.second, Config::TileID::GROUND);

        m_shrinesCollected = m_previousGameState.shrinesCollected;
        m_visitedShrinesList = m_previousGameState.visitedShrinesSnapshot;

        // Đánh dấu đã khôi phục xong
        m_hasPreviousGameState = false;

        // Quay lại game
        m_currentState = STATE_PLAY;
        m_nextState = STATE_PLAY;
        m_isFadingOut = false;
        m_isFadingIn = false;
        m_fadeAlpha = 0.0f;

        // Dừng nhạc menu và phát lại nhạc game
        Mix_HaltMusic();
        SoundManager::GetInstance()->PlayMusic("bgm_game");
```

```
        // Xóa lịch sử state để không loop
        while (!m_stateHistory.empty()) m_stateHistory.pop();

        std::cout << "[GAME] Quay lại game thành công!" << std::endl;
    }
    // 3. Nếu đang ở màn hình khác (và không có trạng thái game): Quay về màn
hình trước
    else {
        if (!m_stateHistory.empty()) {
            GameState prevState = m_stateHistory.top();
            m_stateHistory.pop();

            // Chuyển state trực tiếp (không lưu lịch sử để tránh loop)
            m_currentState = prevState;
            m_nextState = prevState;
            m_isFadingOut = false;
            m_isFadingIn = true;
            m_fadeAlpha = 255.0f; // Bắt đầu fade in
        }
    }
}

// Bật/Tắt âm thanh
void GameEngine::ToggleAudio() {
    // Đảo trạng thái: Đang tắt -> Bật, Đang bật -> Tắt
    m_isMuted = !m_isMuted;
    SoundManager::GetInstance()->SetMute(m_isMuted);
    std::cout << "[ÂM THANH] Trạng thái tắt tiếng: " << (m_isMuted ? "BẬT" :
"TẮT") << std::endl;
}

// CÁC HÀM XỬ LÝ SỰ KIỆN ĐẦU VÀO (INPUT HANDLERS)

void GameEngine::HandleStoryInput(SDL_Event& event) {
    if (event.type == SDL_KEYDOWN && event.key.keysym.sym == SDLK_RETURN) {
        SoundManager::GetInstance()->PlaySFX("sfx_click");

        switch (m_currentState) {
            case STATE_INTRO:
                // Intro -> Bắt đầu vào chuỗi Level 1
                m_currentLevelIdx = 1;
                SwitchState(STATE_STORY_NAME);
                break;

            case STATE_STORY_NAME:
                SwitchState(STATE_STORY_SENSEI);
                break;

            case STATE_STORY_SENSEI:
                LoadLevel(m_currentLevelIdx); // Tải dữ liệu bản đồ
                SwitchState(STATE_PLAY);      // Vào chơi
                break;

            case STATE_RESULT_S:
                 // Hạng S -> Chuyển sang màn tiếp theo (hoặc continue sau
vòng 3)
```

```
                NextLevel();
                break;

            case STATE_RESULT_A:
            case STATE_RESULT_LOSE:
                // Thua hoặc Hạng A -> Chơi lại màn hiện tại (bao gồm vòng 3)
                LoadLevel(m_currentLevelIdx);
                SwitchState(STATE_PLAY);
                break;

            case STATE_CONTINUE:
                {
                    // Từ continue -> vào vòng kế tiếp trực tiếp (không cần
NAME/SENSEI)
                    int nextLevel = m_currentLevelIdx + 1;
                    if (nextLevel > (int)m_levelFiles.size()) {
                        SwitchState(STATE_INTRO);
                        break;
                    }

                    m_currentLevelIdx = nextLevel;
                    LoadLevel(m_currentLevelIdx);
                    SwitchState(STATE_PLAY);
                }
                break;

            default: break;
        }
    }
}

void GameEngine::SwitchState(GameState newState) {
    if (m_currentState == newState) return;
    // Lưu trạng thái cũ vào lịch sử để Undo (trừ khi đang Fade)
    if (!m_isFadingOut) {
        m_stateHistory.push(m_currentState);
    }

    // Nếu đang rời khỏi STATE_PLAY sang các màn hình khác, lưu trạng thái
game hiện tại để có thể quay lại bằng phím U
    if (m_currentState == STATE_PLAY && newState != STATE_PLAY) {
        m_hasPreviousGameState = true;
       m_previousGameState.playerGridRow = (int)((m_pPlayer->GetY() - m_pMap-
>GetOffsetY()) / Config::TILE_SIZE);
       m_previousGameState.playerGridCol = (int)((m_pPlayer->GetX() - m_pMap-
>GetOffsetX()) / Config::TILE_SIZE);
        m_previousGameState.currentSteps = m_currentSteps;
        m_previousGameState.shrinesCollected = m_shrinesCollected;
        m_previousGameState.currentLevelIdx = m_currentLevelIdx;
        m_previousGameState.visitedShrinesSnapshot = m_visitedShrinesList;
    }

    // Reset random rank image when bước sang các màn hình hiển thị/cycle mới
    if (newState == STATE_RESULT_S || newState == STATE_RESULT_A || newState
== STATE_RESULT_LOSE ||
        newState == STATE_STORY_NAME || newState == STATE_CONTINUE) {
```

```
        m_randomRankImageID.clear();
    }

    m_nextState = newState;
    m_isFadingOut = true; // Kích hoạt hiệu ứng Fade Out
}

void GameEngine::NextLevel() {
    int nextLevel = m_currentLevelIdx + 1;

    // Không còn level tiếp theo
    if (nextLevel > (int)m_levelFiles.size()) {
        m_isPendingNextLevel = false;
        SwitchState(STATE_INTRO);
        return;
    }

    // Kết thúc vòng 3 với Rank S -> màn Continue rồi chơi vòng 4
    if (m_currentLevelIdx == 3) {
        m_isPendingNextLevel = false;
        SwitchState(STATE_CONTINUE);
        return;
    }

    if (m_currentLevelIdx < 3) {
        // Vòng 1-2: tăng level và đi qua màn tên/sensei
        m_currentLevelIdx = nextLevel;
        m_isPendingNextLevel = false;
        SwitchState(STATE_STORY_NAME);
        return;
    }

    // Vòng 4 trở đi: tăng level và vào thẳng game
    m_currentLevelIdx = nextLevel;
    m_isPendingNextLevel = false;
    LoadLevel(m_currentLevelIdx);
    SwitchState(STATE_PLAY);
}

void GameEngine::Clean() {
    if (m_pMap) delete m_pMap;
    if (m_pPlayer) delete m_pPlayer;
    SDL_DestroyRenderer(m_pRenderer);
    SDL_DestroyWindow(m_pWindow);
    Mix_Quit();
    TTF_Quit();
    IMG_Quit();
    SDL_Quit();
}
```

8. File GameObject.h

```
#pragma once

// THƯ VIỆN HỆ THỐNG
#include <string>
```

```
#include <SDL.h>

// MẪU THIẾT KẾ ĐỐI TƯỢNG THAM SỐ (PARAMETER OBJECT PATTERN)

// STRUCT: LoaderParams (THAM SỐ KHỞI TẠO)
// Giúp việc truyền tham số khởi tạo trở nên gọn gàng và dễ bảo trì
struct LoaderParams {
public:
     LoaderParams(int x, int y, int width, int height, const std::string&
textureID,
                  int numFrames = 8, int animSpeed = 100)
        : m_x(x), m_y(y), m_width(width), m_height(height),
                          m_textureID(textureID),    m_numFrames(numFrames),
m_animSpeed(animSpeed)
    {}

    // Các hàm truy xuất (Getters)
    int GetX() const { return m_x; }
    int GetY() const { return m_y; }
    int GetWidth() const { return m_width; }
    int GetHeight() const { return m_height; }
    std::string GetTextureID() const { return m_textureID; }
    int GetNumFrames() const { return m_numFrames; }
    int GetAnimSpeed() const { return m_animSpeed; }

private:
    int m_x, m_y;
    int m_width, m_height;
    std::string m_textureID;
    int m_numFrames;
    int m_animSpeed;
};

// CLASS: GameObject (LỚP TRỪU TƯỢNG)
class GameObject {
public:
    // Constructor nhận các thông số cơ bản thông qua LoaderParams
    GameObject(const LoaderParams* pParams)
         : m_x((float)pParams->GetX()), // Ép kiểu sang float để di chuyển
mượt
          m_y((float)pParams->GetY()),
          m_width(pParams->GetWidth()),
          m_height(pParams->GetHeight()),
          m_textureID(pParams->GetTextureID()),
          m_currentRow(0),    // Asset dải ngang -> Luôn là hàng 0
          m_currentFrame(0),   // Frame bắt đầu
          m_numFrames(pParams->GetNumFrames()),
          m_animSpeed(pParams->GetAnimSpeed()),
          m_alpha(255)
    {}

    // Hàm hủy ảo (Virtual Destructor)
    virtual ~GameObject() {}

    // Các hàm thuần ảo (Pure Virtual) bắt buộc lớp con phải định nghĩa lại
    virtual void Draw() = 0;
```

```
    virtual void Update() = 0;
    virtual void Clean() = 0;

    // GETTERS & SETTERS

    // Vị trí thực tế
    float GetX() const { return m_x; }
    float GetY() const { return m_y; }

    // Thiết lập vị trí
    void SetPosition(float x, float y) { m_x = x; m_y = y; }

    // Kích thước
    int GetWidth() const { return m_width; }
    int GetHeight() const { return m_height; }

    // Hàm đổi Texture
    // Dùng để đổi từ "idle_down" sang "run_up" khi di chuyển
    void SetTexture(const std::string& id) { m_textureID = id; }

protected:
    // Vị trí
    float m_x, m_y;

    // Kích thước hiển thị
    int m_width, m_height;

    // ID của ảnh trong TextureManager (VD: "idle_down", "run_right")
    std::string m_textureID;

    // Biến Animation
    int m_currentRow;       // Luôn là 0 vì asset dải ngang
    int m_currentFrame;     // Chạy từ 0 đến 7 (8 frames)
    int m_numFrames;        // Tổng số frame (8)
    int m_animSpeed;        // Tốc độ update frame (ms)

    // Độ mờ (Alpha)
    int m_alpha;
};
```

9. File Map.h

```
#pragma once

// THƯ VIỆN HỆ THỐNG
#include <string>
#include <vector>

// THƯ VIỆN DỰ ÁN
#include "Core/Config.h"   // Để lấy kích thước TILE_SIZE

// STRUCT: MapPoint (TỌA ĐỘ LƯỚI)
struct MapPoint {
    int row; // Chỉ số hàng
    int col; // Chỉ số cột
```

```
    // Hàm khởi tạo (Constructor)
    MapPoint(int r = 0, int c = 0) : row(r), col(c) {}

    // // Toán tử so sánh
    // bool operator==(const MapPoint& other) const {
    //     return row == other.row && col == other.col;
    // }
};

// CLASS: Map (HỆ THỐNG BẢN ĐỒ)
class Map {
public:
    Map();
    ~Map();

    // Tải bản đồ từ file .txt
    void LoadMap(const std::string& path);

    // Vẽ bản đồ lên màn hình
    void DrawMap();

    // CÁC HÀM TRUY VẤN & THIẾT LẬP (GETTERS & SETTERS)

    // Lấy ID của ô tại vị trí cụ thể (0: Đường đi, 1: Không khí, 2: Trận
Nhãn)
    int GetTileID(int row, int col) const;

    // Cập nhật ID của ô (dùng khi thu thập trận nhãn hoặc hoàn tác)
    void SetTileID(int row, int col, int id);

    // Lấy điểm xuất phát của nhân vật
    MapPoint GetStartPoint() const { return m_startPoint; }

    // Lấy danh sách tất cả các trận nhãn (Shrines)
    const std::vector<MapPoint>& GetShrines() const { return m_shrines; }

    // Lấy kích thước bản đồ
    int GetRows() const { return m_rows; }
    int GetCols() const { return m_cols; }

    // Lấy offset (khoảng cách từ góc màn hình tới vị trí vẽ bản đồ)
    int GetOffsetX() const { return m_offsetX; }
    int GetOffsetY() const { return m_offsetY; }

    // Tính tỷ lệ phóng tự động dựa trên kích thước map
    float GetRenderScale() const;

private:
    // Ma trận lưu trữ bản đồ (Vector 2 chiều)
    std::vector<std::vector<int>> m_mapMatrix;

    // Vị trí xuất phát của nhân vật
    MapPoint m_startPoint;

    // Danh sách tọa độ các trận nhãn (TileID = 2)
    std::vector<MapPoint> m_shrines;
```

```
    // Kích thước bản đồ hiện tại
    int m_rows;
    int m_cols;

    // Offset để căn giữa bản đồ trên màn hình
    int m_offsetX;
    int m_offsetY;
};
```

10.File Map.cpp

```
#include "Entities/Map.h"
#include "Graphics/TextureManager.h"
#include "Core/GameEngine.h" // Để truy cập bộ kết xuất đồ họa (Renderer)
#include <fstream>
#include <iostream>

// HÀM KHỞI TẠO & HỦY (CONSTRUCTOR & DESTRUCTOR)

Map::Map() : m_rows(0), m_cols(0), m_offsetX(0), m_offsetY(0) {
    // Vị trí mặc định
    m_startPoint = {0, 0};
}

Map::~Map() {}

// XỬ LÝ DỮ LIỆU (DATA PROCESSING)

void Map::LoadMap(const std::string& path) {
    std::ifstream file(path);
    if (!file.is_open()) {
        std::cerr << "[LỖI] Không thể mở file bản đồ: " << path << std::endl;
        return;
    }

    // 1. Đọc kích thước bản đồ và tổng trận nhãn
    int totalShrines = 0;
    file >> m_rows >> m_cols >> totalShrines;

    if (m_rows <= 0 || m_cols <= 0) {
        std::cerr << "[LỖI NGHIÊM TRỌNG] Kích thước bản đồ không hợp lệ!" <<
std::endl;
        return;
    }

    // Thay đổi kích thước ma trận
    m_mapMatrix.resize(m_rows);
    for (int i = 0; i < m_rows; i++) {
        m_mapMatrix[i].resize(m_cols);
    }

    // Xóa bộ nhớ đệm cũ
    m_shrines.clear();

    m_startPoint = {0, 0};
```

```
    /// 2. Đọc dữ liệu Grid (KÝ TỰ -> ID)
    char tileChar;
    for (int row = 0; row < m_rows; row++) {
        for (int col = 0; col < m_cols; col++) {
            // Đọc từng ký tự, bỏ qua khoảng trắng/xuống dòng
            file >> tileChar;

            int tileID = Config::TileID::GROUND; // Mặc định là đất (0)

            switch (tileChar) {
                case '.': // Đất
                    tileID = Config::TileID::GROUND; // 0
                    break;

                case '#': // Chân không (Air)
                    tileID = Config::TileID::AIR;   // 1
                    break;

                case 'S': // Trận Nhãn (Shrine)
                    tileID = Config::TileID::SHRINE; // 2
                    break;

                default:
                    // Ký tự lạ -> coi như đất
                    tileID = Config::TileID::GROUND;
                    break;
            }

            // Xử lý logic đặc biệt
            if (tileID == Config::TileID::SHRINE) { // Mã 2: Trận nhãn
                m_shrines.push_back(MapPoint(row, col));
            }

            // Lưu vào ma trận
            m_mapMatrix[row][col] = tileID;
        }
    }

    file.close();
    std::cout << "[HỆ THỐNG] Đã tải bản đồ: " << path << " [" << m_rows << "x" << m_cols << "]" << std::endl;
    std::cout << "[THÔNG TIN] Số trận nhãn phát hiện: " << m_shrines.size() << std::endl;

    // Tính toán offset để căn giữa bản đồ trên màn hình, chừa margin đáy
    int mapWidth = m_cols * Config::TILE_SIZE;   // Chiều rộng toàn bộ bản đồ (pixel)
    int mapHeight = m_rows * Config::TILE_SIZE;  // Chiều cao toàn bộ bản đồ (pixel)
    int availableHeight = Config::SCREEN_HEIGHT - Config::MAP_BOTTOM_MARGIN;

    m_offsetX = (Config::SCREEN_WIDTH - mapWidth) / 2;    // Căn giữa theo trục X
    m_offsetY = (availableHeight - mapHeight) / 2;        // Căn giữa theo trục Y trong vùng còn lại
```

```
    if (m_offsetY < 0) m_offsetY = 0;                    // Không để map lệch
âm
}

// KẾT XUẤT ĐỒ HỌA (RENDERING)

void Map::DrawMap() {
    int tileID = 0;

    SDL_Renderer* pRenderer = GameEngine::GetInstance()->GetRenderer();

    const float S = GetRenderScale(); // Tính tự động theo kích thước map

    // Tính offset căn giữa theo tỷ lệ phóng
    int mapWidth = m_cols * Config::TILE_SIZE;
    int mapHeight = m_rows * Config::TILE_SIZE;
    int centerOffsetX = static_cast<int>((Config::SCREEN_WIDTH - mapWidth *
S) / 2.0f);
        int   centerOffsetY   =   static_cast<int>((Config::SCREEN_HEIGHT   -
Config::MAP_BOTTOM_MARGIN - mapHeight * S) / 2.0f);
    if (centerOffsetY < 0) centerOffsetY = 0; // Không để lệch âm khi scale
lớn

    // Dùng tileset.png
    // Mapping:
    // ID 0 (Ground) -> Frame 0 (Xanh lá)
    // ID 1 (Wall)   -> Frame 1 (Xám)
    // ID 2 (Shrine) -> Frame 2 (Xanh dương)

    for (int row = 0; row < m_rows; row++) {
        for (int col = 0; col < m_cols; col++) {
            tileID = m_mapMatrix[row][col];

            // Tọa độ vẽ trên màn hình (áp dụng offset để căn giữa)
                int  x  =  static_cast<int>(col  *  Config::TILE_SIZE  *  S)  +
centerOffsetX;
                int  y  =  static_cast<int>(row  *  Config::TILE_SIZE  *  S)  +
centerOffsetY;

            // VẼ BẰNG PHƯƠNG THỨC DRAW FRAME (CẮT TỪ TILESET)

            if (tileID >= 0) {
                TextureManager::GetInstance()->DrawFrameScaled(
                    "tileset",          // ID của Texture
                    x, y,               // Vị trí vẽ trên màn hình
                    Config::TILE_SIZE,  // Kích thước frame nguồn
                    Config::TILE_SIZE,  // Kích thước frame nguồn
                    S,                  // Tỷ lệ phóng đích
                    0,                  // Hàng số 0 trong tileset
                    tileID,             // Cột (Frame) tương ứng với ID
                    pRenderer           // Bộ kết xuất đồ họa
                );
            }
        }
    }
}
```

```
// TRUY XUẤT THÔNG TIN (DATA ACCESS)

int Map::GetTileID(int row, int col) const {
    // Kiểm tra biên (Bounds Check) để tránh lỗi tràn mảng và crash game
    if (row >= 0 && row < m_rows && col >= 0 && col < m_cols) {
        return m_mapMatrix[row][col];
    }
    return Config::TileID::AIR; // Nếu truy vấn ngoài phạm vi bản đồ, coi như
là ô không khí
}

void Map::SetTileID(int row, int col, int id) {
    if (row >= 0 && row < m_rows && col >= 0 && col < m_cols) {
        m_mapMatrix[row][col] = id;
    }
}

float Map::GetRenderScale() const {
    // Tính tỷ lệ phóng để map vừa khít màn hình với margin dưới 68px
    int mapWidth = m_cols * Config::TILE_SIZE;
    int mapHeight = m_rows * Config::TILE_SIZE;

    // Chiều cao sẵn có (trừ margin dưới)
    int availableHeight = Config::SCREEN_HEIGHT - Config::MAP_BOTTOM_MARGIN;

    // Scale để vừa với chiều rộng và chiều cao có sẵn
    float scaleX = (Config::SCREEN_WIDTH * 0.9f) / mapWidth;
    float scaleY = (float)availableHeight / mapHeight;

    // Chọn scale nhỏ hơn để không vượt quá vùng có sẵn
    float scale = (scaleX < scaleY) ? scaleX : scaleY;

     // Giới hạn tối thiểu 0.5 (không giới hạn tối đa để map lớn vẫn vừa
margin)
    if (scale < 0.5f) scale = 0.5f;

    return scale;
}
```

11.File Player.h

```
#pragma once

#include "Entities/GameObject.h"
#include "Core/Config.h"

// ENUM: ĐỊNH HƯỚNG DI CHUYỂN
enum class PlayerDirection {
    DOWN  = 0,
    UP    = 1,
    LEFT  = 2,
    RIGHT = 3
};

// CLASS: Player
```

```
class Player : public GameObject {
public:
    // Hàm khởi tạo sử dụng LoaderParams từ lớp cha
    Player(const LoaderParams* pParams);

    ~Player() override;

    // Vẽ nhân vật lên màn hình
    void Draw() override;

    // Cập nhật logic nhân vật (Input -> Move -> Animation)
    void Update() override;

    // Dọn dẹp tài nguyên (nếu có)
    void Clean() override;

    // Thiết lập lại vị trí nhân vật
    void SetPosition(float x, float y);

private:
    // Hàm xử lý đầu vào bàn phím
    void HandleInput();

    // Cập nhật hoạt ảnh
    void UpdateAnimation();

    // Xử lý di chuyển vật lý
    void HandleMovement(float dt);

    // Xử lý tương tác với ô gạch (tiles)
    void ProcessTileInteraction();

    // Hiệu ứng rơi xuống vực
    void Falling();

    // Vận tốc (Pixel/Giây)
    float m_velX, m_velY;

    // Trạng thái hướng hiện tại
    PlayerDirection m_currentDir;

    // Lưu hướng cuối cùng khi dừng lại (để hiển thị Idle đúng hướng)
    PlayerDirection m_lastDir;

    // Cờ đánh dấu đang di chuyển hay đứng yên
    bool m_isMoving;

    // Di chuyển theo từng ô: trạng thái và đích đến
    bool m_isMovingToTile;
    float m_targetX;
    float m_targetY;

    // Cờ đánh dấu đang trong quá trình rơi
    bool m_isFalling;

    // Thời gian đã trôi qua kể từ lúc bắt đầu rơi
```

```
    float m_fallTimer;

    // Tốc độ rơi (Co giãn nhỏ dần hoặc tụt xuống theo trục Y)
    const float FALL_DURATION = 1.5f; // Rơi trong 1.5 giây thì chuyển màn
hình thua

    // Tọa độ lúc bắt đầu rơi
    float m_fallStartY;

    // Hàm kiểm tra xem game đã thắng chưa (từ GameEngine)
    bool IsGameWon() const;
};
```

12.File Player.cpp

```
#include "Entities/Player.h"
#include "Core/GameEngine.h"
#include "Graphics/TextureManager.h"
#include "Audio/SoundManager.h"
#include "Entities/Map.h"
#include <cmath> // Dùng cho các phép toán vị trí
#include <algorithm> // std::min

// HÀM KHỞI TẠO & HỦY (CONSTRUCTOR & DESTRUCTOR)

Player::Player(const LoaderParams* pParams)
    : GameObject(pParams),
      m_velX(0.0f),
      m_velY(0.0f),
      m_currentDir(PlayerDirection::DOWN),
      m_lastDir(PlayerDirection::DOWN),
      m_isMoving(false),
      m_isMovingToTile(false),
      m_targetX(0.0f),
      m_targetY(0.0f),
      m_isFalling(false),
      m_fallTimer(0.0f),
      m_fallStartY(0.0f)
{}

Player::~Player() { Clean(); }

void Player::Update() {
    // Lấy khoảng thời gian Delta Time từ Engine
    float dt = GameEngine::GetInstance()->GetDeltaTime();

    // TRƯỜNG HỢP 1: ĐANG RƠI XUỐNG VỰC (FALLING STATE)
    if (m_isFalling) {
        m_fallTimer += dt;

        // 1. Hiệu ứng mờ dần
        // Tính Alpha dựa trên % thời gian đã trôi qua
        float progress = m_fallTimer / FALL_DURATION; // 0.0 -> 1.0
        if (progress > 1.0f) progress = 1.0f;

        m_alpha = (int)(255 * (1.0f - progress)); // Giảm từ 255 về 0
```

```
        if (m_alpha < 0) m_alpha = 0;

        // 2. Hiệu ứng tụt xuống
        // Rơi xuống khoảng 1 ô (64px) trong suốt quá trình
        m_y = m_fallStartY + (progress * 50.0f);

        // 3. Kiểm tra kết thúc
        if (m_fallTimer >= FALL_DURATION) {
            // Chuyển sang màn hình thua
            GameEngine::GetInstance()->SwitchState(STATE_RESULT_LOSE);
        }

        // Khi đang rơi, không xử lý Input hay Animation nữa
        return;
    }

    // TRƯỜNG HỢP 2: TRẠNG THÁI BÌNH THƯỜNG (NORMAL STATE)
    HandleInput();          // 1. Đọc bàn phím
    HandleMovement(dt);     // 2. Di chuyển vật lý & Tương tác ô
    UpdateAnimation();      // 3. Cập nhật hình ảnh (Idle/Run)
}

// XỬ LÝ ĐẦU VÀO (INPUT HANDLING)
void Player::HandleInput() {
    // Nếu đang di chuyển tới một ô, bỏ qua input mới để đảm bảo mượt
    if (m_isMovingToTile) {
        m_isMoving = true;
        return;
    }

    // Mặc định: không di chuyển cho tới khi nhận phím
    m_isMoving = false;

    // LẤY TRẠNG THÁI BÀN PHÍM HIỆN TẠI
    const Uint8* currentKeyStates = SDL_GetKeyboardState(NULL);

    // XỬ LÝ LOGIC DI CHUYỂN (MOVEMENT LOGIC)
                    if         (currentKeyStates[SDL_SCANCODE_UP]         ||
currentKeyStates[SDL_SCANCODE_W]) {
        m_currentDir = PlayerDirection::UP;
        m_isMoving = true;
        m_isMovingToTile = true;
        m_targetX = m_x;
        m_targetY = m_y - Config::TILE_SIZE;
    }
             else      if      (currentKeyStates[SDL_SCANCODE_DOWN]      ||
currentKeyStates[SDL_SCANCODE_S]) {
        m_currentDir = PlayerDirection::DOWN;
        m_isMoving = true;
        m_isMovingToTile = true;
        m_targetX = m_x;
        m_targetY = m_y + Config::TILE_SIZE;
    }
             else      if      (currentKeyStates[SDL_SCANCODE_LEFT]      ||
currentKeyStates[SDL_SCANCODE_A]) {
        m_currentDir = PlayerDirection::LEFT;
```

```
        m_isMoving = true;
        m_isMovingToTile = true;
        m_targetX = m_x - Config::TILE_SIZE;
        m_targetY = m_y;
    }
            else      if      (currentKeyStates[SDL_SCANCODE_RIGHT]      ||
currentKeyStates[SDL_SCANCODE_D]) {
        m_currentDir = PlayerDirection::RIGHT;
        m_isMoving = true;
        m_isMovingToTile = true;
        m_targetX = m_x + Config::TILE_SIZE;
        m_targetY = m_y;
    }

    // Nếu đang di chuyển, cập nhật hướng nhìn cuối cùng
    if (m_isMoving) {
        m_lastDir = m_currentDir;
    }
}

// XỬ LÝ DI CHUYỂN VẬT LÝ
void Player::HandleMovement(float dt) {
    // Di chuyển mượt tới ô mục tiêu nếu có
    if (m_isMovingToTile) {
        float step = Config::PLAYER_SPEED * dt;
        float dx = m_targetX - m_x;
        float dy = m_targetY - m_y;

        if (std::fabs(dx) > 0.0f) {
                float  moveX  =  (dx  >  0  ?  1.0f  :  -1.0f)  *  std::min(step,
std::fabs(dx));
            m_x += moveX;
        }
        if (std::fabs(dy) > 0.0f) {
                float  moveY  =  (dy  >  0  ?  1.0f  :  -1.0f)  *  std::min(step,
std::fabs(dy));
            m_y += moveY;
        }

        // Đã tới mục tiêu -> Snap chính xác vào tâm ô và ghi nhận bước đi
        if (std::fabs(m_targetX - m_x) < 0.001f && std::fabs(m_targetY - m_y)
< 0.001f) {
            m_x = m_targetX;
            m_y = m_targetY;
            m_isMovingToTile = false;
            m_isMoving = false;
            GameEngine::GetInstance()->OnPlayerMove();
            // Sau khi hoàn thành bước, xử lý tương tác ô
            ProcessTileInteraction();
            // Lưu trạng thái sau khi tương tác ô để Undo khôi phục đúng các
ô đã thu thập
            GameEngine::GetInstance()->SaveState();
        }
    }

    // Lấy kích thước Map hiện tại và offset
```

```
    Map* pMap = GameEngine::GetInstance()->GetMap();
    int mapWidth = pMap->GetCols() * Config::TILE_SIZE;
    int mapHeight = pMap->GetRows() * Config::TILE_SIZE;
    int offsetX = pMap->GetOffsetX();
    int offsetY = pMap->GetOffsetY();

    // KIỂM TRA BIÊN BẢN ĐỒ (MAP BOUNDARIES CHECK)
    // So sánh với vị trí của bản đồ (có tính offset)
    bool isOutLeft   = (m_x < offsetX);
    bool isOutTop    = (m_y < offsetY);
    bool isOutRight  = (m_x + Config::TILE_SIZE > offsetX + mapWidth);
    bool isOutBottom = (m_y + Config::TILE_SIZE > offsetY + mapHeight);

    if (isOutLeft || isOutTop || isOutRight || isOutBottom) {
        // Kích hoạt hiệu ứng rơi
        Falling();
        return;
    }
}

void Player::ProcessTileInteraction() {
    // Tính toán TÂM (Center) của nhân vật
    int centerX = (int)(m_x + Config::TILE_SIZE / 2);
    int centerY = (int)(m_y + Config::TILE_SIZE / 2);

    // Lấy thông tin bản đồ và offset
    Map* pMap = GameEngine::GetInstance()->GetMap();
    int offsetX = pMap->GetOffsetX();
    int offsetY = pMap->GetOffsetY();

    // Quy đổi ra tọa độ lưới (Grid Coordinates) - Trừ offset
    int gridCol = (centerX - offsetX) / Config::TILE_SIZE;
    int gridRow = (centerY - offsetY) / Config::TILE_SIZE;

    // Lấy thông tin ô trên bản đồ
    int tileID = pMap->GetTileID(gridRow, gridCol);

    // XỬ LÝ LOGIC TƯƠNG TÁC VỚI Ô LƯỚI (TILE INTERACTION LOGIC)

    // TRƯỜNG HỢP 1: TRẬN NHÃN (SHRINE - ID 2)
    if (tileID == Config::TileID::SHRINE) {
        // Gọi Engine xử lý: Cộng điểm, Phát âm thanh
        GameEngine::GetInstance()->OnShrineVisited(gridRow, gridCol);
    }
    // TRƯỜNG HỢP 2: KHÔNG KHÍ (AIR) Đi vào ô này sẽ RƠI
    else if (tileID == Config::TileID::AIR) {
        // Kích hoạt trạng thái rơi ngay lập tức
        Falling();
    }
}

void Player::Falling() {
    if (m_isFalling) return;

    m_isFalling = true;
    m_fallTimer = 0.0f;
```

```
    m_fallStartY = m_y; // Ghi nhớ độ cao để làm hiệu ứng tụt xuống

    // Phát âm thanh thất bại
    SoundManager::GetInstance()->PlaySFX("sfx_lose");
}

// HỆ THỐNG ANIMATION (VISUALS)
void Player::UpdateAnimation() {
    // 1. Xác định hành động (Action Prefix)
    std::string action = m_isMoving ? "run_" : "idle_";

    // 2. Xác định hướng (Direction Suffix)
    std::string direction = "down"; // Mặc định

    // Nếu đang chạy, dùng hướng hiện tại. Nếu đứng yên, dùng hướng cuối cùng
(LastDir).
    PlayerDirection dirToUse = m_isMoving ? m_currentDir : m_lastDir;

    switch (dirToUse) {
        case PlayerDirection::UP:    direction = "up"; break;
        case PlayerDirection::DOWN:  direction = "down"; break;
        case PlayerDirection::LEFT:  direction = "left"; break;
        case PlayerDirection::RIGHT: direction = "right"; break;
        default: break;
    }

    // 3. Ghép chuỗi để tạo Texture ID
    // Ví dụ: "run_" + "left" = "run_left"
    std::string newTextureID = action + direction;

    // Cập nhật vào GameObject
    SetTexture(newTextureID);

    // 4. Tính toán Frame Animation
    // Công thức: (Thời gian hệ thống / Tốc độ) % Tổng số frame
    // Giúp animation chạy đều bất kể FPS.
    m_currentFrame = int(((SDL_GetTicks() / m_animSpeed) % m_numFrames));
}

// VẼ HÌNH (RENDERING)
void Player::Draw() {
    // Lấy Renderer
    SDL_Renderer* pRenderer = GameEngine::GetInstance()->GetRenderer();
    Map* pMap = GameEngine::GetInstance()->GetMap();

    // [HIỆU ỨNG RƠI] Cập nhật độ trong suốt (Alpha Mod)
    // Nếu m_alpha < 255, nhân vật sẽ mờ đi.
    // Lấy texture hiện tại từ Manager để set Alpha
              SDL_Texture*      tex      =      TextureManager::GetInstance()-
>GetTexture(m_textureID);
    if (tex) SDL_SetTextureAlphaMod(tex, (Uint8)m_alpha);

    const float S = pMap ? pMap->GetRenderScale() : 1.0f; // Scale theo bản
đồ

    // Tính offset căn giữa map (khớp với Map::DrawMap, có chừa margin đáy)
```

```
    int mapCols = pMap ? pMap->GetCols() : 0;
    int mapRows = pMap ? pMap->GetRows() : 0;
    int mapWidth = mapCols * Config::TILE_SIZE;
    int mapHeight = mapRows * Config::TILE_SIZE;
    int centerOffsetX = static_cast<int>((Config::SCREEN_WIDTH - mapWidth *
S) / 2.0f);
        int   centerOffsetY   =   static_cast<int>((Config::SCREEN_HEIGHT   -
Config::MAP_BOTTOM_MARGIN - mapHeight * S) / 2.0f);
    if (centerOffsetY < 0) centerOffsetY = 0;

    // Lấy tâm ô theo tọa độ thế giới (m_x, m_y đang là góc trên trái ô +
offset map)
    float worldTileCenterX = m_x + Config::TILE_SIZE * 0.5f - (pMap ? pMap-
>GetOffsetX() : 0);
    float worldTileCenterY = m_y + Config::TILE_SIZE * 0.5f - (pMap ? pMap-
>GetOffsetY() : 0);

    // Quy đổi sang tọa độ màn hình đã scale và căn giữa
       int   screenCenterX   =   static_cast<int>(worldTileCenterX   *   S)   +
centerOffsetX;
       int   screenCenterY   =   static_cast<int>(worldTileCenterY   *   S)   +
centerOffsetY;

    // Kích thước sprite sau khi scale theo map
    int scaledWidth = static_cast<int>(m_width * S);
    int scaledHeight = static_cast<int>(m_height * S);

    // Vẽ sao cho nhân vật luôn nằm giữa ô
    TextureManager::GetInstance()->DrawFrameScaled(
        m_textureID,
        screenCenterX - scaledWidth / 2,
        screenCenterY - scaledHeight / 2,
        m_width,
        m_height,
        S,
        m_currentRow,
        m_currentFrame,
        pRenderer
    );

    // Khôi phục độ trong suốt về 255 sau khi vẽ
    if (tex) SDL_SetTextureAlphaMod(tex, 255);
}

// Thiết lập lại vị trí nhân vật
void Player::SetPosition(float x, float y) {
    m_x = x;
    m_y = y;
    m_targetX = x;
    m_targetY = y;
    m_isMovingToTile = false;

    // Khôi phục toàn bộ trạng thái khi được dịch chuyển (Teleport)
    m_isFalling = false;
    m_alpha = 255;
    m_fallTimer = 0.0f;
```

```
    m_velX = 0;
    m_velY = 0;

    // Khôi phục animation về trạng thái đứng yên hướng xuống
    m_currentDir = PlayerDirection::DOWN;
    m_lastDir = PlayerDirection::DOWN;
    m_isMoving = false;
}

void Player::Clean() {}
```

13.File TextureManager.h

```
#pragma once

// THƯ VIỆN HỆ THỐNG
#include <string>
#include <map>
#include <SDL.h>
#include <SDL_ttf.h>

// MẪU THIẾT KẾ SINGLETON

class TextureManager {
public:
    // Lấy thể hiện duy nhất của TextureManager
    static TextureManager* GetInstance();

    // Hủy và giải phóng instance khi tắt game
    static void DestroyInstance();

    // 2. QUẢN LÝ TÀI NGUYÊN (RESOURCE MANAGEMENT)

    // Tải hình ảnh (Load image/texture)
       bool  Load(const  std::string&  fileName,  const  std::string&  id,
SDL_Renderer* pRenderer);

    // Tải phông chữ (Load font)
     bool LoadFont(const std::string& fileName, const std::string& id, int
fontSize);

    // Xóa một texture cụ thể khỏi bộ nhớ
    void ClearTexture(const std::string& id);

    // Xóa toàn bộ tài nguyên (Texture & Font)
    void Clean();

    // 3. KẾT XUẤT ĐỒ HỌA (RENDERING)

    // Vẽ toàn bộ Texture lên màn hình (Dùng cho hình nền)
     void Draw(const std::string& id, int x, int y, int width, int height,
SDL_Renderer* pRenderer);

    // Vẽ một phần của Texture (Dùng cho Animation/Spritesheet)
    void DrawFrame(const std::string& id, int x, int y, int width, int height,
int currentRow, int currentFrame, SDL_Renderer* pRenderer);
```

```
    // Vẽ một phần của Texture với tỷ lệ phóng (chỉ phóng kích thước vẽ, không thay đổi kích thước cắt nguồn)
    void DrawFrameScaled(const std::string& id, int x, int y, int srcWidth, int srcHeight, float scale, int currentRow, int currentFrame, SDL_Renderer* pRenderer);

    // 4. Hàm truy xuất Texture (Getter)
        SDL_Texture*    GetTexture(const    std::string&    id)    {    return m_textureMap[id]; }

private:
    // Hàm khởi tạo riêng tư (Private Constructor - Singleton)
    TextureManager();
    ~TextureManager();

    // Thể hiện duy nhất (Instance)
    static TextureManager* s_Instance;

     // Cấu trúc Map lưu trữ Texture: Khóa (Key) = Tên ID chuỗi, Giá trị (Value) = Con trỏ SDL_Texture*
    std::map<std::string, SDL_Texture*> m_textureMap;

    // Cấu trúc Map lưu trữ Font: Khóa = Tên ID, Giá trị = Con trỏ TTF_Font*
    std::map<std::string, TTF_Font*> m_fontMap;
};
```

14.TextureManager.cpp

```
#include "Graphics/TextureManager.h"
#include <SDL_image.h>
#include <iostream>

// Khởi tạo con trỏ Singleton là null ban đầu
TextureManager* TextureManager::s_Instance = nullptr;

TextureManager* TextureManager::GetInstance() {
    // Chỉ tạo instance khi được gọi lần đầu tiên
    if (s_Instance == nullptr) s_Instance = new TextureManager();
    return s_Instance;
}

void TextureManager::DestroyInstance() {
    if (s_Instance != nullptr) {
        s_Instance->Clean(); // Dọn dẹp tài nguyên trước khi hủy đối tượng
        delete s_Instance;
        s_Instance = nullptr;
    }
}

TextureManager::TextureManager() {}

TextureManager::~TextureManager() { Clean(); }

void TextureManager::Clean() {
    // 1. Giải phóng toàn bộ Texture
```

```
    // Duyệt qua từng phần tử trong Map và hủy Texture
          for    (auto    const&    [id,    texture]    :    m_textureMap)
SDL_DestroyTexture(texture);
    m_textureMap.clear(); // Xóa sạch danh sách định danh

    // 2. Giải phóng toàn bộ Font
    for (auto const& [id, font] : m_fontMap) TTF_CloseFont(font);
    m_fontMap.clear();

     std::cout << "[ĐỒ HỌA] Bộ nhớ TextureManager đã được dọn dẹp." <<
std::endl;
}

bool TextureManager::Load(const std::string& fileName, const std::string& id,
SDL_Renderer* pRenderer) {
    // Kiểm tra xem texture đã được tải chưa
    if (m_textureMap.find(id) != m_textureMap.end()) return true;

    SDL_Surface* pTempSurface = IMG_Load(fileName.c_str());
    if (pTempSurface == nullptr) {
        std::cerr << "[LỖI] TextureManager::Load - Không thể tải hình ảnh: "
<< fileName
                  << " | Lỗi SDL_Image: " << IMG_GetError() << std::endl;
        return false;
    }

        SDL_Texture*   pTexture   =   SDL_CreateTextureFromSurface(pRenderer,
pTempSurface);
    SDL_FreeSurface(pTempSurface);

    if (pTexture == nullptr) {
        std::cerr << "[LỖI] TextureManager::Load - Không thể tạo texture từ
surface: " << fileName
                  << " | Lỗi SDL: " << SDL_GetError() << std::endl;
        return false;
    }

    m_textureMap[id] = pTexture;
    std::cout << "[TÀI NGUYÊN] Đã tải Texture: " << id << " (" << fileName <<
")" << std::endl;

    return true;
}

void TextureManager::ClearTexture(const std::string& id) {
    // Tìm iterator của texture cần xóa
    auto it = m_textureMap.find(id);

    // Nếu tìm thấy
    if (it != m_textureMap.end()) {
        SDL_DestroyTexture(it->second); // Hủy texture
        m_textureMap.erase(it);         // Xóa khỏi danh sách quản lý
        std::cout << "[TÀI NGUYÊN] Đã xóa Texture: " << id << std::endl;
    }
}
```

```
// 3. QUẢN LÝ PHÔNG CHỮ (FONT MANAGEMENT)

bool TextureManager::LoadFont(const std::string& fileName, const std::string&
id, int fontSize) {
    if (m_fontMap.find(id) != m_fontMap.end()) return true;

    // Mở file phông chữ với kích thước cụ thể
    TTF_Font* pFont = TTF_OpenFont(fileName.c_str(), fontSize);
    if (pFont == nullptr) {
        std::cerr << "[LỖI] TextureManager::LoadFont - Không thể tải phông
chữ: " << fileName << " | Lỗi TTF: " << TTF_GetError() << std::endl;
        return false;
    }

    m_fontMap[id] = pFont;
    std::cout << "[TÀI NGUYÊN] Đã tải Font: " << id << " (" << fontSize <<
"pt)" << std::endl;
    return true;
}

// 4. KẾT XUẤT ĐỒ HỌA (RENDERING)

void TextureManager::Draw(const std::string& id, int x, int y, int width, int
height, SDL_Renderer* pRenderer) {
    // Nếu texture chưa tải, thoát để tránh lỗi crash
    if (m_textureMap.find(id) == m_textureMap.end()) {
        std::cerr << "[CẢNH BÁO] Không tìm thấy ID Texture: " << id <<
std::endl;
        return;
    }

    SDL_Rect srcRect;  // Vùng lấy ảnh gốc
    SDL_Rect destRect; // Vùng vẽ trên màn hình

    // Lấy toàn bộ kích thước ảnh gốc
    srcRect.x = 0;
    srcRect.y = 0;
    srcRect.w = destRect.w = width;
    srcRect.h = destRect.h = height;

    // Vị trí đích
    destRect.x = x;
    destRect.y = y;

    // Gọi lệnh vẽ của SDL
    SDL_RenderCopy(pRenderer, m_textureMap[id], &srcRect, &destRect);
}

void TextureManager::DrawFrameScaled(const std::string& id, int x, int y, int
srcWidth, int srcHeight, float scale, int currentRow, int currentFrame,
SDL_Renderer* pRenderer) {
    if (m_textureMap.find(id) == m_textureMap.end()) return;

    SDL_Rect srcRect;
    SDL_Rect destRect;
```

```
    srcRect.x = srcWidth * currentFrame;
    srcRect.y = srcHeight * currentRow;
    srcRect.w = srcWidth;
    srcRect.h = srcHeight;

    destRect.x = x;
    destRect.y = y;
    destRect.w = static_cast<int>(srcWidth * scale);
    destRect.h = static_cast<int>(srcHeight * scale);

    SDL_RenderCopy(pRenderer, m_textureMap[id], &srcRect, &destRect);
}

void TextureManager::DrawFrame(const std::string& id, int x, int y, int width,
int height, int currentRow, int currentFrame, SDL_Renderer* pRenderer) {
    // Nếu texture chưa load, thoát để tránh Crash
    if (m_textureMap.find(id) == m_textureMap.end()) return;

    SDL_Rect srcRect;
    SDL_Rect destRect;

    // XỬ LÝ CẮT SPRITESHEET (SPRITESHEET CLIPPING LOGIC)

    // Dịch khung nhìn sang phải theo số thứ tự frame
    srcRect.x = width * currentFrame;

    // Dịch khung nhìn xuống dưới (ảnh là 1 hàng nên row = 0)
    srcRect.y = height * currentRow;

    // Kích thước ô cắt
    srcRect.w = width;
    srcRect.h = height;

    // Vị trí vẽ
    destRect.x = x;
    destRect.y = y;
    destRect.w = width;  // Vẽ đúng bằng kích thước 1 frame
    destRect.h = height;

    SDL_RenderCopy(pRenderer, m_textureMap[id], &srcRect, &destRect);
}
```

15.File main.cpp

```
#include "Core/GameEngine.h"
#include "Core/Config.h"
#include <iostream>
#include <stdexcept>
#include <windows.h>

const int TARGET_FPS = 60;
const int FRAME_DELAY = 1000 / TARGET_FPS;

int main(int argc, char* argv[]) {
    // Cấu hình Console Tiếng Việt (UTF-8) để debug
    SetConsoleOutputCP(CP_UTF8);
```

```
    SetConsoleCP(CP_UTF8);

      std::cout << "================================================" << std::endl;
     std::cout << "[HỆ THỐNG] Đang khởi động Game KY MON THAN TOC..." << std::endl;
    std::cout << "[HỆ THỐNG] Tốc độ khung hình mục tiêu: " << TARGET_FPS << " FPS" << std::endl;
      std::cout << "================================================" << std::endl;

    Uint32 frameStartTimestamp;
    int frameDuration;

    GameEngine* pGameEngine = GameEngine::GetInstance();

    try {
        if (!pGameEngine->Init("Ky Mon Than Toc",
                               SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
                                  Config::SCREEN_WIDTH, Config::SCREEN_HEIGHT, true)) {
            throw std::runtime_error("Khởi tạo Game thất bại!");
        }

        std::cout << "[HỆ THỐNG] Khởi tạo Game thành công. Đang bắt đầu vòng lặp game..." << std::endl;

        while (pGameEngine->IsRunning()) {
            frameStartTimestamp = SDL_GetTicks();

            pGameEngine->HandleEvents();
            pGameEngine->Update();
            pGameEngine->Render();

            frameDuration = SDL_GetTicks() - frameStartTimestamp;
            if (FRAME_DELAY > frameDuration) {
                SDL_Delay(FRAME_DELAY - frameDuration);
            }
        }

    } catch (const std::exception& e) {
         std::cerr << "[LỖI NGHIÊM TRỌNG] Ngoại lệ chưa được xử lý: " << e.what() << std::endl;
        GameEngine::DestroyInstance();
        return -1;
    } catch (...) {
        std::cerr << "[LỖI NGHIÊM TRỌNG] Đã xảy ra ngoại lệ không xác định!" << std::endl;
        GameEngine::DestroyInstance();
        return -1;
    }

    std::cout << "[HỆ THỐNG] Đang tắt chương trình..." << std::endl;
    GameEngine::DestroyInstance();
    std::cout << "[HỆ THỐNG] Tạm biệt!" << std::endl;
    return 0; }
```