CISC 322
A3: Architectural Enhancement report
December 2, 2024

Architectural Enhancement of ScummVM

Group4:
Lorem Ipsum

Authors:
Dima Zaichenko (22bh17@queensu.ca)

Joshua Yarussky (21jy71@queensu.ca)

Roman Rodchenkov (21rr69@queensu.ca)

Mohammed Sarhat(19ms16@queensu.ca)

Charles Letellier(21cl106@queensu.ca)

Patrick Favret(21phf1@queensu.ca)

Table of contents

Abstract

This report examines the enhancement of ScummVM's architecture to introduce standardized and streamlined multiplayer functionality. ScummVM reimplements classic game engines, enabling retro games to run on modern platforms. Key components include GUI, Backends, Common, and Engines are analyzed for limitations, revealing the need for a new Multiplayer Component to be added in the top-level architecture. This component is built on a subsystem that has a Session Manager, Sync Engine, Network Manager, Input Router and Rule Manager, these facilitate session management, synchronization, and network communication for the system. The enhancement improves scalability, evolvability, and maintainability while addressing risks such as security vulnerabilities, compatibility issues, and performance challenges through modular design and extensive testing. Despite limitations like game-specific feasibility, the proposed solution encapsulates multiplayer functionality in a dedicated module, enabling collaborative gameplay and ensuring adaptability for future needs.

Introduction

ScummVM, short for Script Creation Utility for Maniac Mansion Virtual Machine, is a robust open-source software that reimplements classic game engines, allowing vintage adventure games to be played on modern platforms. It has gained recognition for its ability to preserve gaming history by supporting hundreds of titles from the 80s and 90s, including games originally developed for proprietary or obsolete systems. ScummVM is built on a modular architecture that emphasizes flexibility, maintainability, and portability. This architecture has enabled ScummVM to thrive as a community-driven project, continuously evolving to support new platforms and engines.

Despite its success, ScummVM lacks built-in functionality for multiplayer gameplay, a feature that could extend its appeal and functionality. While ScummVM has excelled in reimplementing game engines for single-player experiences, adding support for multiplayer gaming would represent a significant enhancement. Currently, the responsibility for implementing multiplayer functionality falls entirely on the original game or engine. This inconsistency not only increases the development effort for integrating multiplayer games but also creates a barrier for expanding ScummVM's library with multiplayer-capable titles.

The proposed enhancement aims to address this limitation by introducing standardized multiplayer support into ScummVM. This document explores two potential approaches to implementing multiplayer functionality. The first approach involves modifying existing components like the Backend, Engines, and GUI to support networking capabilities. This approach assumes that multiplayer synchronization and logic remain the responsibility of individual game engines. The second approach suggests creating a dedicated Multiplayer Component, a new top-level subsystem designed to centralize and streamline multiplayer functionality, including session management, networking, input handling, and synchronization.

Both approaches are analyzed in detail, highlighting their advantages and challenges. While modifying existing components builds on ScummVM's modular design, it places significant responsibilities on individual game engines, potentially complicating their integration. Conversely, a dedicated multiplayer component offers a standardized architecture for multiplayer features, simplifying the addition of new multiplayer games but requiring substantial architectural changes.

This document also discusses the implications of adding multiplayer functionality, including potential effects on performance, maintainability, and testability. A structured SAAM (Software Architecture Analysis Method) analysis evaluates how the proposed enhancement aligns with ScummVM's functional and nonfunctional requirements, emphasizing backward compatibility and minimal disruption to single-player experiences. Furthermore, the document provides an overview of the existing architecture, identifies the gaps that the enhancement would address, and details the impacted directories and files.

Finally, the document outlines testing strategies and associated risks for the enhancement, including performance challenges, security concerns, and compatibility issues. Lessons learned throughout this exploration offer insights into the complexities of enhancing legacy systems like ScummVM, underscoring the importance of balancing flexibility, feasibility, and maintainability in architectural decisions.

By implementing multiplayer support, ScummVM could significantly expand its capabilities, making classic games more engaging for modern audiences. This enhancement not only preserves the nostalgia of vintage games but also creates new opportunities for players to enjoy these games together, fostering a sense of community and collaboration.

Proposed Enhancement

The proposed enhancement to improve ScummVM is to streamline and standardize multiplayer for supporting games. Currently ScummVM has no built-in way of allowing multiplayer to be seamlessly added; it is currently up to the individual game to implement multiplayer in whichever way they choose to. This means that changes to the architecture have to be made to allow for this new, standardized, functionality. Below, we will discuss two approaches to solving this problem.

Approach 1: Modify Backends to Support Multiplayer

This approach involves heavy modification of the Backend, Engine, and Utilities components to allow support for multiplayer. In a typical multiplayer game, multiplayer functionality is implemented at a low level (assuming networked multiplayer, and not local/splitscreen multiplayer - those games should already work with ScummVM without significant modification). Additionally, since this approach does not involve the creation of any new components, and purely modifying existing components, synchronization of data flow would be purely up to the game or game engine it is running on - meaning the games must already support multiplayer with this approach.

The Backend component would require networking support, specifically, socket-based communication such as TCP and UDP for reliable and low-latency communication. Additional

APIs for network primitives such as opening, closing, and accepting a connection, as well as actually sending and receiving data over the connection. In addition, event handling for network events should need consideration. All these proposed changes would need to also be mirrored in the OSystemAPI class and implemented in a platform-specific way for each platform ScummVM supports, meaning that it will be no small task to update the existing Backends component.

The Engines component would also (potentially) need modifications. If a multiplayer game's engine does not make use of the aforementioned OSystemAPI methods, it would have to now, as the game engines would need to send and receive game-specific state changes, handle network events, and take care of game synchronization to ensure the game plays correctly.

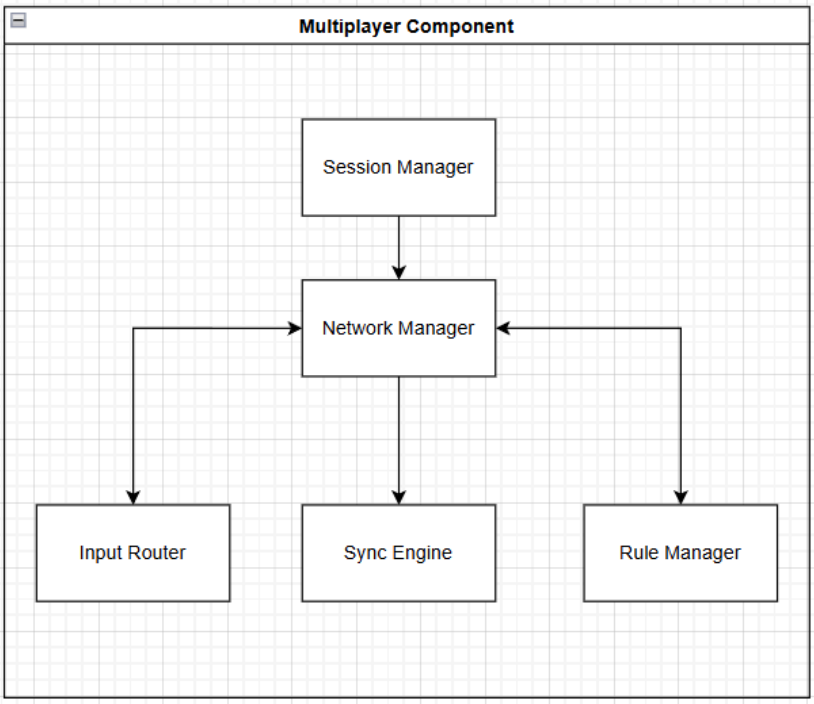
Finally, the GUI would need updates to integrate the multiplayer functionality for supporting games and to facilitate/ease the connection between players. This approach of not creating any new high-level components and instead modifying existing systems is good because it builds upon the existing modular structure of ScummVM, leverages the Backend's existing abstraction functionality, and puts the responsibility of synchronization and conflict resolution on game engines. Conversely, this approach may put too much responsibility on the game engines, which are already a very complicated component, and make it much harder to evolve the product by adding new game engines, however, the changes to the backend to facilitate network communication would most likely have to be implemented even if this approach is not taken.

Approach 2: Create new Multiplayer Component

This approach involves creating an entirely separate multiplayer component and adding it to the top level architecture. This multiplayer component would have a sub system that would hold both server components and client components, running a client/server architecture style allowing an easy connection for multiple users. Using one style for the networking would allow a standardized multiplayer architecture for any game that would be added to scummVM.

The subsystem as shown in figure 1, consists of a Session manager which is in charge of the storing and tracking session data and being middleware between the GUI and the server components. The Network Manager is responsible for maintaining a secure connection and reliable connection between users, it sends and receives data allowing users to be on the same page when playing the game,the Network Manager would include protocols for connection security and would handle reconnection logic in case of network failures. Finally, it coordinates the Input Router, Sync Engine, and Rule Manager, ensuring they work together to maintain synchronization and enforce game rules. The Input Router processes and directs player inputs to the appropriate game engines or multiplayer components, ensuring that all user actions are accurately transmitted to the server. It also validates inputs to prevent desynchronization and ensures fairness during gameplay. The Sync Engine ensures that the game state remains consistent across all clients and the server by updating the current state of the game, this ensures that there is a seamless connection between the users and the server ensuring little to no packet loss and desync across users. The Rule Manager enforces game-specific rules and logic, ensuring that all player actions adhere to predefined game mechanics. It validates moves, prevents cheating, and manages event triggers, such as conditions for scoring or completing a level. The modular design of the Rule Manager allows easy adaptation to different games with ScummVM's library.

All of these things put together would help games added to scummVM that have multiplayer as an easy way of establishing a secure multiplayer experience that is streamlined and standardized across the whole system simplifying development.



[Figure 1] Multiplayer component subsystem

SAAM Analysis

Stakeholder	Notable Non-Functional Requirements
Players	**Performance:** Multiplayer gameplay should have reduced latency. There should also be high responsiveness. **Usability:** Hosting, joining, and managing multiplayer sessions should have an intuitive interface. **Reliability:** There should be no disruptions within multiplayer sessions. There should be a smooth and consistent experience.
Developers	**Maintainability:** Complexity should be minimized. The enhancement should be easy to debug. **Testability:** The system should allow targeted testing of multiplayer features without overly impacting unrelated components. **Modifiability:** Changes to multiplayer features should not interfere with single-player functionalities.
Game Publishers	**Compatibility:** Multiplayer functionality should not negatively impact single-player games, or features.

Community and Ecosystem	**Scalability:** Multiplayer features should support a wide range of games and platforms. **Accessibility:** The systems should provide an accessible and standardized multiplayer experience.

[Table 1] Stakeholder Consideration of Non-Functional Requirements

Attribute	Approach 1: Modify Existing Components	Approach 2: Creating A New Multiplayer Component
Performance	**Moderate:** Performance varies entirely depending on how individual engineers function. Peer-to-peer or client-server designs may introduce latency issues.	**High:** A centralized engine ensures efficient and consistent state synchronization across all clients.
Safety	**Moderate:** Relies heavily on individual game engines to manage network safety, making errors more likely.	**High:** A centralized design ensures robust mechanisms for handling network failures and user interactions.
Maintainability	**Low:** Developers must modify the Backend, Engines, and GUI for every game This increases maintenance complexity, and development time.	**High:** A modular multiplayer component simplifies maintenance, keeping changes isolated.
Robustness	**Low:** A lack of standardization across engineers could lead to failures or desynchronization.	**Moderate:** Centralized handling ensures robust multiplayer logic, but may still need specific adjustments for games.
Testability	**Moderate:** This approach supports scalability for each game, but lacks a unified design. This allows for large-scale multiplayer more challenging.	**High:** Testing is possible due to the isolation of multiplayer features in a single component.

[Table 2] Impact of Approach 1 vs Approach 2 on Non-Functional Requirements

Approach 2, which introduces a dedicated Multiplayer Component, is superior to Approach 1 due to its centralized and modular design. By isolating multiplayer functionality into a single component, Approach 2 ensures better maintainability, testability, and scalability. It allows testing of multiplayer features without affecting other components. The introduction of a centralized engine guarantees consistent state synchronization across all clients, addressing

performance concerns more effectively than Approach 1, which relies on individual game engines for networking and synchronization.

Current State

This section examines the existing components of ScummVM that are relevant to the proposed enhancement. It then identifies any new components or alterations that are required to successfully implement the enhancement.

1. Current relevant components to the proposed enhancement:

- GUI
  - Provides a user interface for selecting and configuring games.
  - Handles user interaction within games and ScummVM as a whole.
- Backends
  - Interacts with the underlying platform to handle operating system specific operations allowing for supported games to be played on a wide array of modern systems.
- Common
  - Includes many useful utilities such as serialization. Serialization converts the current game state into a format that can be transferred over a network, a key feature in any multiplayer game.
- Engines
  - Handles game specific scripts and gameplay logic for supported games including any multiplayer functionality if it is already implemented in the original game.

2. What is Missing:

- GUI
  - The GUI component lacks user interface elements such as buttons or menus for the user to initiate or join multiplayer sessions.
  - The GUI component must integrate with the session manager to facilitate multiplayer functionality, enabling players to host, join, and manage multiplayer sessions.
- Backends
  - The Backends component lacks the infrastructure for establishing network connections such as TCP and UDP sockets nor does it handle reconnection logic in the event that the connection is lost.
  - These tasks would rely on the Network Manager within the Multiplayer Component which will handle secure connections and enable access to server components and systems.
- Engines
  - Currently, the Engines component does not support synchronization for multiplayer across all supported games.

    - To solve this, the Engines will need to interface with the Synchronization Engine (Sync Engine) in the Multiplayer Component in order to support the synchronization required for multiplayer.
- Multiplayer Component
    - The Multiplayer Component is missing from the current architecture and is essential to the successful implementation of the proposed enhancement. It is responsible for filling in the gaps in game state synchronization, network communication and session management.
    - The Multiplayer component consists of several subcomponents:
        - Session Manager: manages the hosting, joining, and tracking of multiplayer sessions.
        - Network Manager: Ensures secure and reliable connections between users and handles reconnection logic.
        - Input Router: Processes and validates user input, ensuring they are applied in the correct order.
        - Sync Engine: Ensures a consistent and synchronized game state across all users and resolves desynchronization issues.
        - Rule Manager: Enforces game-specific rules and resolves conflicts such as determining the outcome when multiple players perform an action simultaneously.

Effects of enhancement

Performance
Implementing multiplayer functionality introduces network communication overhead, which could affect the performance of ScummVM. Real-time synchronization of game states across multiple clients requires continuous data exchange, potentially increasing CPU and memory usage. Network latency and packet loss might also impact the responsiveness of games. However, by designing the Multiplayer Component efficiently—using optimized networking protocols and minimizing unnecessary data transmission—we can mitigate these performance impacts. While there may be a slight decrease in performance due to the overhead, the enhancement provides significant new functionality that justifies the trade-off. Single-player games should remain unaffected, ensuring that the core performance of ScummVM is maintained.

Evolvability
Adding the Multiplayer Component improves the evolvability of ScummVM. By modularizing multiplayer functionality into a dedicated component, we make it easier to add support for new multiplayer games in the future. The standardized networking and synchronization mechanisms mean that individual game engines do not need to implement their own multiplayer logic from scratch.

This modular design also allows for easier updates and improvements to multiplayer functionality without affecting other parts of the system. Future enhancements, such as adding support for new networking protocols or improving synchronization algorithms, can be made within the Multiplayer Component, making ScummVM more adaptable to future requirements.

Testability and Maintainability

The enhancement positively impacts testability by isolating multiplayer functionality within a dedicated component. This modularity allows for targeted testing of multiplayer features without needing to test the entire system. We can create test cases specifically for the Session Manager, Network Manager, and other subcomponents.

Maintainability is also improved, as changes to multiplayer features can be made within the Multiplayer Component without impacting other components. This separation of concerns reduces complexity and makes the codebase easier to understand and maintain over time. Developers can update or refactor multiplayer code without affecting unrelated parts of the system, enhancing the overall maintainability of ScummVM.

Impacted directories and files

In order to implement the proposed enhancements for implementing multiplayer into ScummVM, various files, and directories must be altered to fit the updated architecture and subsystems/components. The following directories and files are impacted by the addition of multiplayer into ScummVM's systems.

Backend/

Network support and handling would be required to implement low-level networking functionalities like TCP/UDP socket creation, connection management, and event handling. This would allow for methods such as openConnection, terminateConnection, sending and receiving (for network events), etc.

Engines/

The Engines/ directory would be impacted to include certain game engine-specific files and modifications to handle synchronization and state updates received from the Sync Engine, which will interface with every other engine for consistent state synchronization.

GUI/

The GUI/ directory would need a new file to provide user interface elements for hosting, joining, and managing multiplayer sessions, as well as updates to existing GUI files to integrate new menus or buttons for multiplayer setup.

Common/

The Common/ directory will require a new file containing functionality to enable serialization and deserialization of game states for transmission over the network. It would also require a network utilities file for things such as data formatting, validation, and packet handling.

Multiplayer/

Perhaps the most robust adjustment would be the addition of a Multiplayer/ directory, containing new files such as the session manager, used for managing multiplayer sessions, as well as the hosting and joining functionality. It would also require the network manager file, to handle

connection-related logic, the Sync Engine, as previously discussed, and routers to validate user input.

Build/

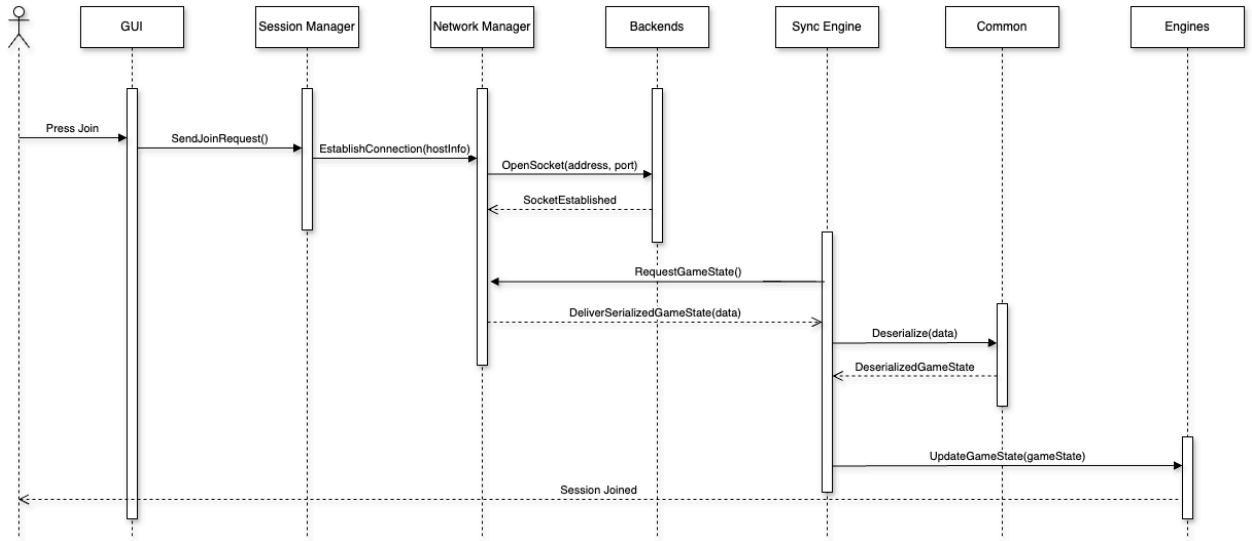
The build directory will simply be updated to include multiplayer-related libraries and extensions.

Interactions with other features

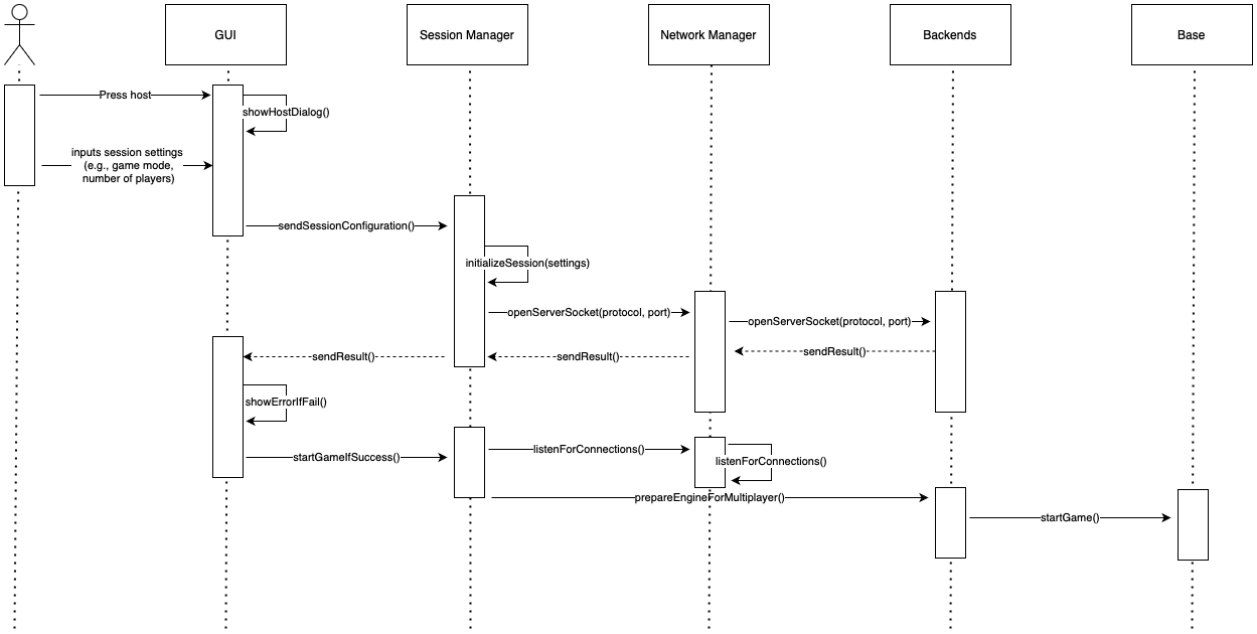
Using approach two, the multiplayer subsystem interacts with existing top-level components such as GUI, Backend, Common, and Engines. The Backend connects directly to the Network Manager, enabling quick access to server components and systems. The GUI interacts with the Session Manager, which serves as middleware between the frontend and backend. The Session Manager provides users with control to either host or join a server, a critical feature for multiplayer as it allows players to decide who they play with. The Sync Engine integrates with the Engines component to ensure seamless game state synchronization between users on the server. It handles real-time updates, maintaining consistency in game logic, events, and player interactions across all connected players. The Input Router connects to the Network Manager and vise-versa. Input Router receives user input data from the client side, which is then processed to server-side data and sent back to the network manager to ensure all users see the results of those inputs in real time. The Rule Manager connects directly to the Engines component to enforce game-specific rules and validate player actions. It ensures that all gameplay behaviors comply with predefined mechanics, such as move validation, scoring conditions, or event triggers. To maintain consistency across all clients, the Rule Manager also interacts with the Sync Engine, ensuring that rule enforcement aligns with the synchronized game state. Through its connection to the Network Manager, the Rule Manager monitors and validates networked game actions, ensuring they are legitimate and compliant with both the game's rules and the overall ScummVM system.

Use Cases

Use Case 1: Joining a Multiplayer Session



Use Case 2: Hosting a Multiplayer Session



Test Plan

Testing is a critical component in validating multiplayer functionality for ScummVM, ensuring reliability, performance, and integration across various games. The primary objectives include ensuring consistent functionality across all multiplayer games, minimizing latency during gameplay to enhance user experience, and addressing compatibility challenges across game backends.

Low-risk scenarios involve non-critical components of the multiplayer system that while potentially impacting the user experience, do not disrupt gameplay. These include chat functionality, player profile synchronization, or minor latency spikes. The goal is to ensure the system maintains stable gameplay and mitigates the impact of non-critical failures. One key method for testing low-risk scenarios is through connectivity tests. Connectivity tests verify that multiplayer sessions can be established under various network conditions by simulating peer-to-peer and client-server setups to assess robustness. To perform these tests, a simulation of minor network issues such as temporary interruptions, or limited bandwidth can be simulated to evaluate the system's ability to maintain functionality and ensure a smooth experience.

Medium-risk scenarios involve failures in subsystems that affect the multiplayer experience; however, it does not render it unplayable. These scenarios include network connectivity issues, delayed synchronization, or unexpected client disconnections. The goal is to ensure that the system can maintain stability and restore normal gameplay. One key method for testing medium-risk scenarios is through failure recovery tests. Failure recovery tests evaluate the system's ability to handle unexpected disconnections, including reconnection attempts. To perform these tests, scenarios such as temporary network outages or client-server interruptions are simulated.

High-risk scenarios involve critical subsystem failures that significantly disrupt or disable gameplay. This includes server crashes, protocol handler errors, or hardware failures. The goal is to ensure that the system halts safely under these conditions, preserving user data and providing clear feedback to players. One key method for testing high-risk scenarios is through load tests. Load tests simulate server-client interactions with multiple participants to evaluate scalability and identify bottlenecks under high stress. To perform these tests, the number of concurrent users is gradually increased beyond the expected capacity to determine the system's maximum supported load and measure its response.

Risks

Security Risks

One of the primary concerns is security. Introducing network capabilities opens up the system to potential external threats such as hacking, unauthorized access, and data breaches. Without proper security measures like encryption, authentication, and secure protocols, malicious users could exploit vulnerabilities to gain unauthorized access to users' systems or manipulate game sessions. Ensuring that the Network Manager and related components handle data securely is crucial to protect both the users and the integrity of the games.

Compatibility Issues

Since ScummVM supports a wide range of classic games, integrating multiplayer functionality might cause compatibility issues with some games. Not all games were originally designed with multiplayer in mind, and attempting to retrofit multiplayer capabilities could result in unexpected behavior or game instability. Thorough testing is required to ensure that multiplayer enhancements do not negatively affect single-player experiences or break existing functionality.

Performance Risks

Through the addition of multiplayer functionality in ScummVM, there is a risk for potential performance challenges– particularly in maintaining low latency, and optimizing resource usage. Real-time state synchronization can place a significant load on the system, especially in resource-constrained environments. Ensuring the system is optimized for scalability is crucial to avoid breaking existing functionality.

Limitations and Lessons learned

One limitation of our proposed enhancement is that not every game that ScummVM supports is suitable for multiplayer functionality. For example, a story-driven single player game may not benefit from or even feasibly support multiplayer functionality. Designing a universal multiplayer component that works seamlessly with all supported titles would be ambitious and very complex to implement. Even in games that could benefit from multiplayer functionality, the diversity of game mechanics and requirements introduces significant challenges, such as synchronization, session management, and input handling, which may need to be handled on a per game basis.

Additionally, the concrete implementation of this component could vary significantly from the conceptual architecture. Practical implementation of the multiplayer component may reveal unanticipated interactions or dependencies. For instance, some subcomponents may need to merge or are missing altogether. Implementing this enhancement would likely be an iterative process that requires flexibility and openness to deviating from the conceptual architecture.

This project has taught us the importance of balancing flexibility, feasibility, and long-term maintainability when implementing an architectural enhancement. Encapsulating multiplayer functionality into a dedicated component promotes scalability and modifiability in the future. However, it also requires a team of well-rounded experts in fields such as networking, synchronization and software architecture. All in all, these three reports have offered us a comprehensive exploration of software architecture and the complexities of working with large systems. By reflecting on these lessons, we have come to appreciate the unique challenges and opportunities that arise when enhancing a legacy system like ScummVM.

Data dictionary

Term	**Definition**	**Type**
ScummVM	An open source software that reimplements game engines to allow classic adventure games to run on modern platforms.	Software
OSystemAPI	A layer within the system architecture that deals with platform-specific details. Allows game engines to run on multiple platforms without issues.	Subsystem
TCP	A commonly used networking protocol that prioritizes	Networking Protocol

	reliability and accuracy of order.	
UDP	A commonly used networking protocol that prioritizes speed over reliability.	Networking Protocol

Naming conventions

ScummVM	Script Creation Utility for Maniac Mansion Virtual Machine
Sync Engine	Synchronization Engine
GUI	Graphical User Interface
API	Application Programming Interface
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

References

ScummVM, "ScummVM GitHub Repository," [Online]. Available: https://github.com/scummvm/scummvm. [Accessed: Nov. 28, 2024].

ScummVM Documentation: ScummVM, "ScummVM Official Documentation," [Online]. Available: https://docs.scummvm.org/. [Accessed: Nov. 30, 2024].