Enhancing ScummVM; Community Engagement

Authors
Harry George – 20327293
Deniz Aydin – 20299521
Ben Hilderman – 20374738
Riley Spavor – 20218837
Kurtis Marinos – 20336452
Jacob Skiba – 20361187

Table of Contents

Group 18

Abstract

This report proposes an **achievement and leaderboard system** to modernize ScummVM while preserving its core mission of retro game emulation. The system rewards players with badges, points, and titles for completing games and achieving milestones, displayed on dynamic leaderboards and personalized profiles.

The proposed system integrates into ScummVM's event-driven architecture, leveraging its Engines, GUI, and Common subsystems. Two implementation strategies are evaluated—Incremental Integration and Modular Overhaul—alongside a Software Architecture Analysis Method (SAAM) analysis to determine the optimal approach. Risks, including performance and compatibility concerns, are identified and mitigated through testing and validation plans.

1. Overview of the Enhancement

1.1 Overview

This enhancement adds an achievement and leaderboard system to ScummVM, modernizing the platform while preserving its mission of retro game preservation. Players earn badges, points, and titles for milestones, enhancing engagement and replayability.

Key features include:

- Player profiles showcasing achievements and progress.
- Dynamic leaderboards filterable by region, game type, and completion speed.
- Seamless integration with ScummVM's existing architecture for compatibility and performance.

This enhancement fosters user engagement and social interaction, aligning ScummVM with modern gaming trends while honoring its legacy.

1.2 Motivation and Benefits

ScummVM's absence of progress tracking, rewards, and social features limits replayability and risks alienating younger audiences, ultimately hindering user retention and broader appeal. The proposed achievement and leaderboard system addresses these gaps by increasing engagement through milestone completion, fostering community with leaderboards that encourage friendly competition, and modernizing the platform to attract new users while retaining its core audience. By integrating seamlessly with ScummVM's architecture, this enhancement enriches the user experience and ensures the platform remains relevant in the evolving gaming landscape.

2. Introduction

2.1 Overview of ScummVM

ScummVM is an open-source platform that allows users to play classic adventure and role-playing games by emulating their original game engines. Initially developed to support LucasArts' SCUMM (Script Creation Utility for Maniac Mansion) engine, ScummVM has expanded to include numerous engines, supporting hundreds of games across various genres. Its mission is to preserve gaming history by ensuring that retro games remain accessible on modern hardware. ScummVM is widely celebrated for its cross-platform compatibility, lightweight architecture, and ease of use, making it a cornerstone of retro game preservation.

2.2 Current System Limitations

Despite its strengths, ScummVM lacks features that engage modern audiences and enhance the user experience. These limitations, identified in the A2 report, include:

1. **Absence of Engagement Features**:
   ScummVM does not offer progress tracking, rewards, or any mechanism to recognize player achievements. This limits the incentive for players to revisit or fully complete games.
2. **Lack of Social Connectivity**:
   The platform operates as a solitary experience, with no tools for players to connect, compare progress, or engage with a broader community.
3. **Modernization Gap**:
   While ScummVM is a powerful tool for retro gaming enthusiasts, its lack of competitive or community-driven features risks alienating younger audiences accustomed to modern gaming environments with achievements, leaderboards, and interactive profiles.

Without these features, ScummVM's ability to retain users and attract new ones is limited, posing challenges to its long-term growth and relevance.

2.3 Rationale for the Proposed Enhancement

To address these gaps, the proposed achievement and leaderboard system introduces modern gaming features while respecting ScummVM's mission of game preservation. By rewarding players for completing games and reaching milestones, the enhancement:

- **Boosts Engagement**:
  Players are incentivized to revisit games, track their progress, and strive for achievements.
- **Fosters Community**:
  Leaderboards create opportunities for friendly competition and social interaction, turning ScummVM into a shared experience.
- **Modernizes the Platform**:
  Introducing these features bridges the gap between classic game nostalgia and contemporary gaming expectations, making ScummVM appealing to broader demographics.

This enhancement seamlessly integrates into ScummVM's architecture, leveraging existing subsystems like Engines, GUI, and Common while introducing new modules such as the Achievement Tracker and Leaderboard Management System. By doing so, it preserves ScummVM's lightweight, event-driven design while enriching the user experience.

3. Proposed Enhancement

3.1 Feature Description

The proposed enhancement introduces a dynamic **achievement and leaderboard system** into ScummVM. This system will reward players with badges, points, and titles for completing games or achieving specific milestones. The new feature integrates deeply with ScummVM's existing event-driven, interpreter-based architecture, ensuring that game progress triggers appropriate updates to the achievement and leaderboard functionalities.

Achievements will be logged and displayed within a player's profile, alongside a dynamically updated leaderboard that showcases progress and rankings. Players can interact with their profiles, explore their accomplishments, and compare performance with others via filters such as region, game type, or completion speed. The leaderboard updates occur in real-time, preserving ScummVM's responsiveness while maintaining compatibility with its existing architecture.

The achievement and leaderboard system leverages ScummVM's core components:

- Game engines, which act as interpreters, generate and transmit events related to game milestones.
- Shared resources from the Common subsystem support achievement data handling and storage.
- Real-time communication between subsystems ensures that notifications, profile updates, and leaderboard adjustments occur seamlessly without disrupting gameplay.

3.2 Use Cases

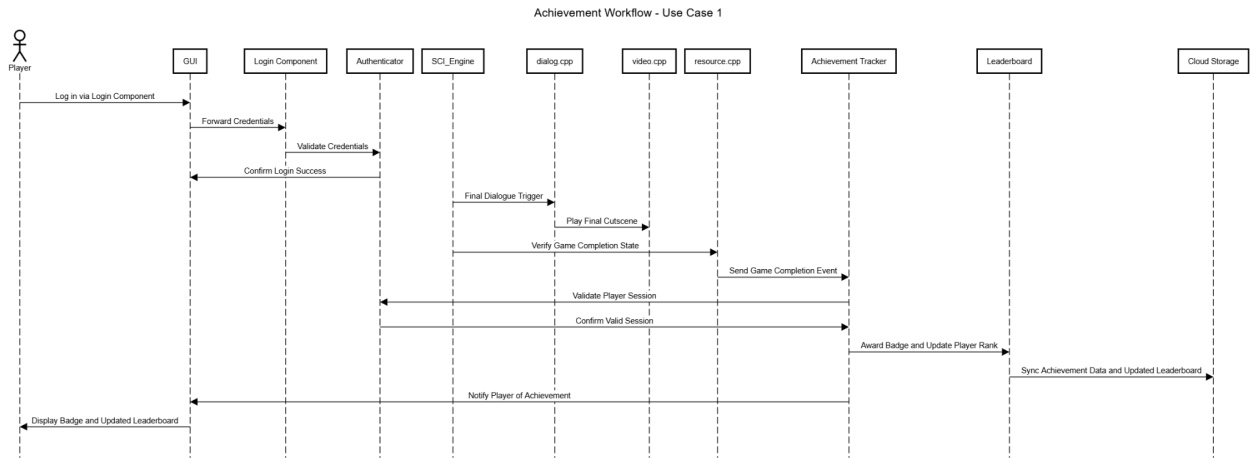
The following use cases illustrate how ScummVM integrates the enhanced workflow, including the new login, achievements, and leaderboard features.

3.2.1 Use Case 1: Achievements Workflow

This use case demonstrates how ScummVM processes an achievement when a player completes a game. The player logs into their account via the Login Component. After successful authentication through the Authenticator, the SCI Engine detects the completion event (e.g., end credits or final dialogue). The event is processed through dialog.cpp and video.cpp, and the SCI Engine sends a game completion event to the Achievement Tracker.

The Achievement Tracker validates the user session with the Authenticator before awarding the player a badge and updating their rank on the Leaderboard. The updated leaderboard data is synced to Cloud Storage to ensure cross-device consistency. Finally, the GUI notifies the player of their new badge and rank while displaying the updated leaderboard.

This workflow highlights the event-driven interaction between the SCI Engine, Achievement Tracker, and Leaderboard, showcasing how ScummVM integrates achievement processing into its architecture.



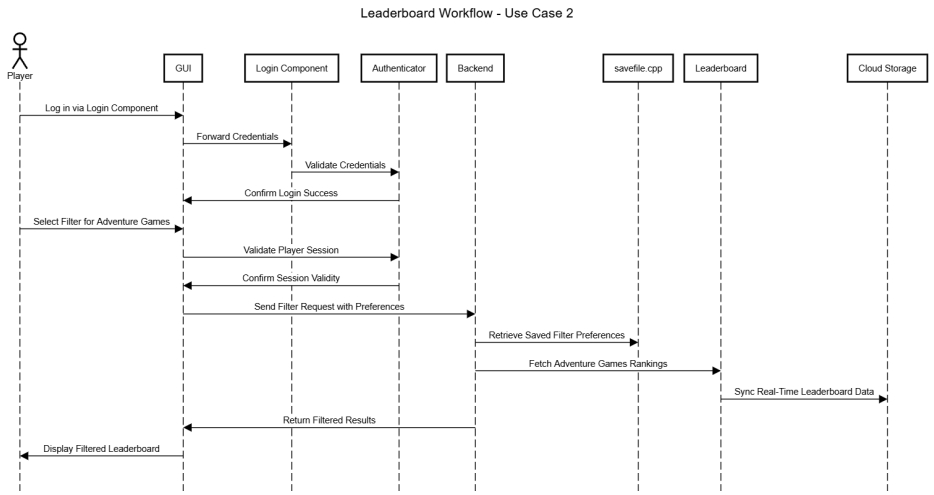
Sequence Diagram 1: This diagram shows a player logging in through the Login Component, which validates their credentials with the Authenticator. Upon game completion, the SCI Engine triggers the achievement workflow, passing events to the Achievement Tracker, which updates the Leaderboard and syncs with Cloud Storage. The GUI displays the player's badge and leaderboard updates.

3.2.2 Use Case 2: Leaderboard Workflow

This use case outlines how a player filters leaderboard results, such as viewing the top-ranked players in the adventure genre. The player logs into ScummVM through the Login Component, with their credentials validated by the Authenticator. Once logged in, the GUI allows the player to select filter options, like genre or friends' rankings.

The GUI validates the player session with the Authenticator and forwards the filter preferences to the backend. The backend retrieves leaderboard data from the Leaderboard, applies the filter, and synchronizes real-time results with Cloud Storage. The filtered leaderboard data is returned to the GUI, which dynamically updates the view for the player.

This workflow highlights the personalized data filtering process, focusing on the interaction between the GUI, Leaderboard, and backend components.



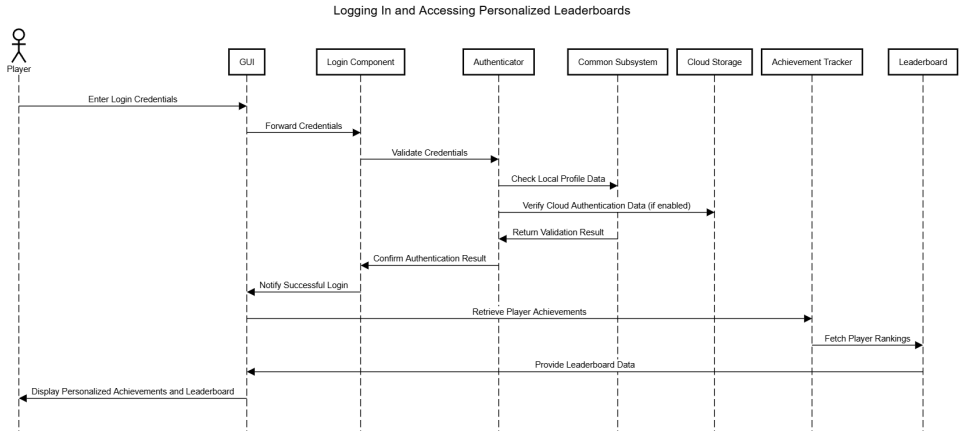
Sequence Diagram 2: The diagram begins with a stick-figure player logging in via the Login Component, followed by session validation through the Authenticator. The player selects filter preferences in the GUI, which queries the backend. The backend retrieves leaderboard data, applies filters, and syncs with Cloud Storage. The GUI displays the filtered results dynamically to the player.

3.2.3 Use Case 3: Logging In and Accessing Personalized Leaderboards

This use case demonstrates the login process for ScummVM and how personalized features like achievements and leaderboards are accessed. The player enters their credentials in the Login Component, which forwards them to the Authenticator. The Authenticator checks local profile data and, if necessary, verifies credentials via Cloud Storage. Upon successful authentication, the GUI retrieves achievements and leaderboard rankings from the Achievement Tracker and Leaderboard.

The GUI dynamically displays the player's personalized badges, achievements, and rankings. If authentication fails, the GUI prompts the player to re-enter their credentials.

This workflow highlights the login process, emphasizing secure authentication and personalized data retrieval.



Sequence Diagram 3: The diagram starts with a stick-figure player logging in via the Login Component. The credentials are validated by the Authenticator, which checks local and cloud-based data. The GUI retrieves achievements and leaderboard rankings from the Achievement Tracker and Leaderboard, displaying personalized data to the player.

Integration with Current Architecture

ScummVM’s event-driven design allows this enhancement to integrate naturally into its existing subsystems. Game engines, such as the SCI Engine, already operate as interpreters of game scripts, generating events in response to user actions. The achievement system will extend this functionality by introducing new event types tied to milestones and completions. These events will be logged and processed using ScummVM's shared resources provided by the Common subsystem.

The GUI will be expanded to include achievement displays and leaderboard views, ensuring compatibility with ScummVM’s current interface. The backend will handle the storage and retrieval of leaderboard data, supporting real-time updates and synchronization. This design respects ScummVM’s decentralized nature, allowing subsystems to communicate bidirectionally without introducing unnecessary coupling.

Design Rationale

This enhancement aligns with ScummVM’s existing architecture by leveraging its decentralized, event-driven communication and shared resource management. The SCI Engine continues to act as the central orchestrator of game-specific logic, dynamically triggering subsystem interactions to support the achievement and leaderboard functionalities. By embedding the feature directly into ScummVM’s interpreter-based design, the system maintains its scalability and extensibility while offering a modernized user experience.

The achievement system reflects ScummVM’s commitment to preserving retro games while engaging a broader audience through community-driven features. By enhancing the user experience without compromising the platform’s architectural integrity, this feature bridges the gap between classic gaming nostalgia and contemporary expectations.

4. Interactions with the Current System

4.1 Impacted Components and Interfaces

The proposed achievement and leaderboard system will seamlessly integrate with ScummVM’s event-driven and interpreter-based architecture by leveraging its existing subsystems. The **Engines subsystem**, including game-specific engines like the SCI Engine, will act as the primary source of achievement events, triggering milestone completions (e.g., finishing a game) through newly introduced event listeners that transmit data to the Achievement Tracker without requiring significant structural changes. The **Common subsystem**, ScummVM’s hub for shared utilities, will handle data storage and retrieval for achievements by incorporating an Achievement Tracker to log events and manage user profiles and leaderboard entries, utilizing existing file I/O utilities to ensure cross-platform compatibility. The **GUI subsystem** will be updated to include an Achievements Screen displaying badges and milestones, a Leaderboard View for filtering and viewing rankings, and notification functionality for new achievements, all consistent with ScummVM’s current themes and layout files. Finally, the **Backend subsystem** will manage local leaderboard storage, incorporating real-time data updates and caching mechanisms to minimize latency, while ensuring scalability for potential future cloud-based leaderboards or save synchronization, aligning with ScummVM’s adaptability goals.

4.2 Architectural Evolution

ScummVM’s architecture will evolve to support the new achievement and leaderboard system while maintaining its core event-driven design principles. Proposed changes include **event propagation**, where the Engines subsystem emits milestone events (e.g., game completion) that are queued and processed asynchronously by the Achievement Tracker in the Common subsystem, minimizing gameplay disruption. **Enhanced data flow** will dynamically transfer achievement data from the Engines subsystem to the Achievement Tracker, which updates local user profiles and leaderboard data accessible via the GUI. For instance, when the SCI Engine triggers a "completion event," the Tracker updates the user’s profile with a badge and adjusts leaderboard rankings, propagating the changes to the GUI for real-time visualization. **GUI extensions** will introduce menus and displays for achievements and leaderboards, consistent with ScummVM’s lightweight interface to ensure usability. Leveraging **concurrency mechanisms** already used for audio playback and game saving, the system will handle updates like leaderboard synchronization without interrupting gameplay. To future-proof the system, **refactoring for scalability** will enable offline leaderboards initially, with the Backend subsystem designed for potential cloud synchronization, aligning with ScummVM’s long-term adaptability goals.

5. Architectural Impact

5.1 High-Level Architecture Changes

The introduction of the achievement and leaderboard system results in notable high-level changes to ScummVM's architecture. These changes enhance functionality while preserving the event-driven design principles and decentralized interactions that define the platform. Two major components are added: the **Achievement Tracker** and the **Leaderboard Management System**.

The Achievement Tracker, integrated within the Common subsystem, will act as a centralized processor for events generated by the Engines subsystem. Whenever a game event, such as a milestone or completion, occurs, the Achievement Tracker will log it and convert it into meaningful achievements. This processed data will then propagate to the GUI for display and to the Leaderboard Management System for ranking updates.

The Leaderboard Management System is introduced as part of the Backend subsystem. It manages local storage of leaderboard data, facilitating real-time updates for user profiles and rankings. By maintaining offline functionality, this system supports scalability for future online features without compromising the existing architecture.

The overall data flow between these components maintains ScummVM's event-driven approach. Events originate in the Engines subsystem, are processed by the Achievement Tracker, and are distributed to the Leaderboard and GUI subsystems for final rendering and interaction. Bidirectional communication between the GUI and Backend subsystems ensures that leaderboard updates are displayed seamlessly.

5.2 Low-Level Architecture Adjustments

At a more detailed level, the proposed enhancement introduces changes across ScummVM's key subsystems. These adjustments ensure the smooth integration of the new features while adhering to existing architectural conventions.

Within the Engines subsystem, new event listeners will be added to detect milestones such as game completions or in-game objectives. For example, the SCI Engine will emit specific events into ScummVM's existing event queues, maintaining the asynchronous flow of gameplay logic.

The Common subsystem will host the Achievement Tracker, extending its shared utilities to handle achievement data. This tracker will log events from the Engines subsystem, process them into badges or rankings, and store the results for later retrieval. Additionally, the Common subsystem's schema will be updated to include new data structures for achievement and leaderboard records, ensuring compatibility with existing save files and cross-platform functionality.

The GUI subsystem will be extended to include new screens for achievements and leaderboards, alongside a notification feature that informs users of their progress. The Achievements Screen will display badges and player milestones, while the Leaderboard View will allow users to filter and explore rankings dynamically. These additions will maintain consistency with ScummVM's established design patterns and theming in the gui/themes/ directory.

In the Backend subsystem, the Leaderboard Management System will introduce caching mechanisms to store frequently accessed data. This ensures efficient leaderboard queries and supports the real-time nature of the feature. The storage and retrieval of leaderboard data will follow the subsystem's existing file-handling conventions, aligning with ScummVM's focus on portability and low overhead.

5.3 Implications on Maintainability, Evolvability, and Performance

The architectural adjustments made to accommodate the achievement and leaderboard system will have significant implications for maintainability, evolvability, and performance.

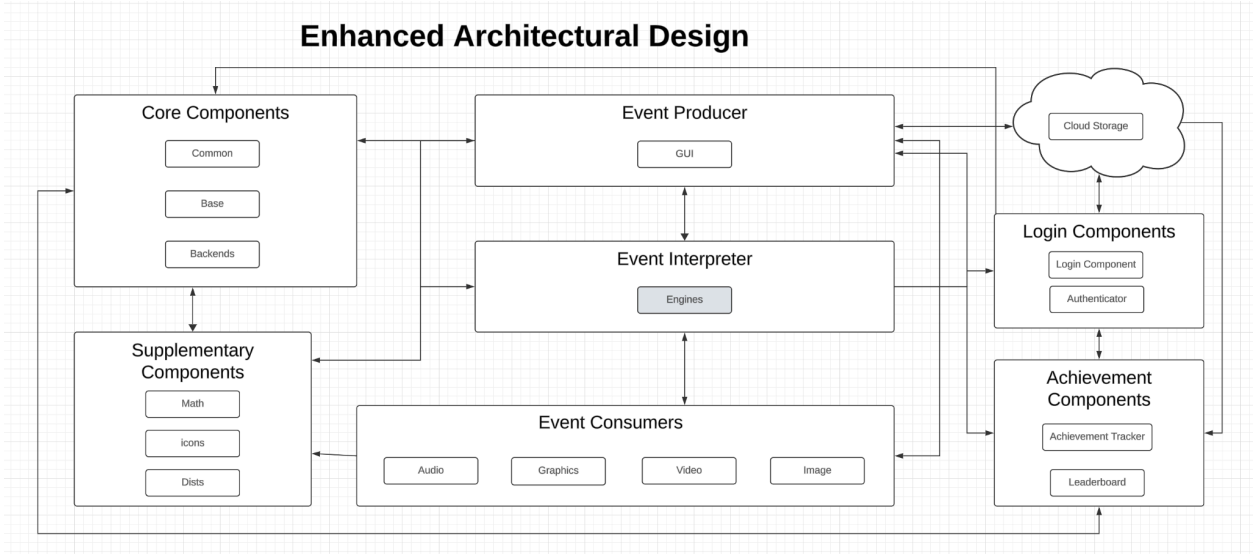
In terms of maintainability, the Achievement Tracker and Leaderboard Management System are designed as cohesive extensions to the Common and Backend subsystems, respectively. This approach minimizes disruption to existing components, making updates and debugging processes straightforward. The reliance on event listeners within Engines

ensures that the enhancement integrates smoothly across ScummVM's diverse library of supported games, avoiding engine-specific dependencies.

Evolvability is enhanced by the design's reliance on shared utilities and standardized event propagation. The system is well-positioned to support future expansions, such as new achievement types, additional filters for leaderboard views, or even online synchronization features. These changes can be introduced incrementally without significant restructuring.

From a performance perspective, the event-driven mechanism ensures that achievement and leaderboard updates do not interfere with gameplay. Real-time updates are supported by caching within the Backend subsystem and asynchronous communication between the GUI and other components. This design minimizes latency while maintaining the system's responsiveness, even under heavy load.

Diagrams



The high-level architecture diagram illustrates the enhanced design of ScummVM, detailing the interaction between its existing subsystems—Engines, Common, GUI, and Backend—and the newly introduced Achievement Tracker and Leaderboard Management System. The core components, which include Common, Base, and Backend modules, form the foundation of the architecture, supporting the Supplementary Components such as Math, Icons, and Distributions. The Event Producer (GUI) facilitates user interaction and generates events, which are processed by the Event Interpreter (Engines). These events are then consumed by the Event Consumers (Audio, Graphics, Video, and Image) to deliver the appropriate output.

The new Login Components, consisting of a Login module and an Authenticator, ensure secure user access, while the Achievement Components, comprising the Achievement Tracker and Leaderboard, manage user progress and competition. Cloud storage integration is utilized to maintain persistence across sessions and enable synchronization of user data. This diagram emphasizes the flow of events within the system, from gameplay interactions to leaderboard updates and user notifications, showcasing a cohesive architecture that seamlessly incorporates new features while maintaining compatibility with ScummVM's existing structure.

6. Alternatives for Realization

The realization of the achievement and leaderboard system can follow one of two potential approaches. Each approach considers ScummVM's event-driven and interpreter-based architecture, ensuring compatibility with the existing system while meeting the design goals outlined in Section 5. Below, we discuss the two alternatives, evaluate their strengths and weaknesses, and recommend the most suitable approach.

6.1 Approach 1: Incremental Integration

The **incremental integration approach** introduces the achievement and leaderboard system into ScummVM in small, manageable phases, leveraging its decentralized architecture to avoid overhauling the entire system at once. The key phases involve developing and integrating the Achievement Tracker to log milestones from the Engines subsystem, expanding the GUI subsystem to include an Achievements Screen for displaying user progress, implementing the Leaderboard Management System within the Backend subsystem (initially supporting offline rankings), and connecting the GUI to the Backend subsystem for leaderboard queries and interaction. This approach offers low risk by minimizing disruption, enables early feedback through independent testing of each phase, and maintains compatibility with ScummVM's event-driven architecture. However, it may result in longer development time and temporary inconsistencies, as some features may not function fully until later phases are implemented.

6.2 Approach 2: Centralized Overhaul

The **centralized overhaul approach** involves designing and implementing the achievement and leaderboard system as a standalone feature before integrating it into ScummVM. This method focuses on creating a self-contained module that interacts with existing subsystems after its development is complete. The implementation process includes developing the Achievement Tracker and Leaderboard Management System as independent modules using mock data, testing the standalone system extensively, and then integrating it into ScummVM by establishing connections with the Engines, Common, GUI, and Backend subsystems. This approach allows for efficient parallel development, simplifies debugging and future updates through separation of concerns, and enables faster deployment upon completion. However, it carries higher initial risk due to delayed integration, which may reveal compatibility issues late in the process, and requires comprehensive testing to ensure seamless functionality across subsystems.

6.3 Comparison of Strategies

The table below summarizes the trade-offs between the two approaches based on key criteria:

Criterion	Incremental Integration	Centralized Overhaul
Development Time	Longer (phased)	Shorter (parallel development)
Risk of System Disruption	Low	Moderate
Compatibility	High (integrates incrementally)	Moderate (tested post-development)
Maintainability	Moderate	High
Scalability	Moderate (adjusted in phases)	High (centralized design)
Early Feedback	Yes	No

6.4 Recommendation

After evaluating both approaches, we recommend **Incremental Integration** as the preferred strategy for implementing the achievement and leaderboard system. This approach aligns better with ScummVM's architectural principles and prioritizes minimizing risk and ensuring compatibility with the platform's extensive library of supported games. By introducing changes gradually, ScummVM can maintain its robust functionality while providing opportunities for iterative testing and feedback.

The centralized overhaul approach, while efficient and scalable, introduces higher risks due to delayed integration. ScummVM's diverse set of subsystems and event-driven interactions make early compatibility testing critical to avoid system-wide disruptions.

7. SAAM Analysis

The ScummVM architecture must support the achievement and leaderboard system in a way that satisfies the needs of various stakeholders while maintaining non-functional requirements (NFRs) such as performance, usability, and scalability.

This section applies the Software Architecture Analysis Method (SAAM) to evaluate the two alternative implementation approaches discussed in Section 6.

7.1 Stakeholder Identification

The achievement system involves several key stakeholders, each with unique interests. Players and gamers who use ScummVM will earn achievements and are primarily interested in easy access to their accomplishments, competitive rankings, and social recognition. ScummVM developers, tasked with implementing and maintaining the feature, prioritize clean, maintainable code that integrates seamlessly with ScummVM's architecture. Community members, including those who may not play games but interact with ScummVM's community, value an aesthetically pleasing and intuitive interface for browsing achievements. Game developers and publishers, as licensing stakeholders, are concerned with ensuring that the achievement system aligns with game licenses and respects the original vision of their games. ScummVM administrators, responsible for moderating community features, require tools to manage abuse, verify achievements, and ensure fair play within the system.

7.2 Non-Functional Requirements (NFRs)

NFR	Stakeholder	Description
Performance	Players, Developers	The system must not significantly slow down gameplay or ScummVM's performance.
Scalability	Developers, Administrators	The system should handle a growing user base and large leaderboards.
Usability	Players, Community Members	The system must be intuitive, with easy navigation of achievements and ranks.
Security	Developers, Admins	Prevent tampering, false achievement claims, and data breaches.
Maintainability	Developers	Should be easy to debug, update, and integrate with ScummVM's existing codebase.
Portability	Players, Developers	Compatible across platforms (Windows, macOS, Linux, mobile devices).
Privacy	Players, Admins	Respect user privacy by allowing opt-in/opt-out of public leaderboards.
Community Moderation Tools	Admins	Provide tools to address abuse (e.g., cheating or harassment).

7.3 Evaluation of Implementation Approaches

Approach 1: A Local System with an Online Leaderboard Integration

- Achievements are tracked locally on the user's device and synchronized with a central server for leaderboard functionality.

NFR	Evaluation
Performance	Local tracking ensures minimal impact on gameplay performance.
Scalability	Requires robust server-side design to handle large-scale leaderboard data.
Usability	Seamless integration with UI to show achievements and leaderboards.
Security	More secure as the central server can validate achievement data.
Maintainability	Requires maintenance of both local code and server-side backend.
Portability	Achievements can be locally synced across platforms.
Privacy	Opt-in system allows users to decide if data is shared publicly.
Moderation Tools	Admin tools can be implemented on the server for effective community management.

Approach 2: Fully Decentralized System

- Achievements and rankings are stored and processed entirely locally, with optional peer-to-peer sharing for public viewing.

NFR	Evaluation
Performance	May impact performance due to local storage and processing of large leaderboard data.
Scalability	Limited scalability as the system relies on individual devices rather than a central server.
Usability	Complex for users to manage; may lack a polished community leaderboard feature.
Security	Vulnerable to tampering, as there's no central authority to validate achievements.
Maintainability	Easier to maintain since no server infrastructure is needed.

Portability	Fully portable as no server dependency exists.
Privacy	Maximized privacy since data is stored locally unless explicitly shared.
Moderation Tools	Difficult to implement as there's no central system to enforce rules or manage reports.

Given this analysis, the best approach would be a local system on an online leaderboard. Using a decentralized system would allow vulnerabilities in temperament of rewards and user's scoring. There's no central authority that would validate completion of achievements. It would also be difficult especially if ScummVM is operating internationally,

7.4 Final Recommendation

Using a local system with an online leaderboard integration would balance both performance and scalability, as achievements are tracked locally but synchronized with a central server. With that, the security is enhanced by allowing server-side validation of achievements, reducing the risk of cheating. This therefore, offers a better user experience by providing centralized leaderboards and intuitive interfaces for browsing ranks. Having a centralized system also supports moderation and privacy, enabling users to opt-in while providing admins tools to ensure fairness. While this approach requires additional maintenance for the backend, and would require ScummVM developers to volunteer their time to "check-in" and moderate, this trade-off ensures a superior and engaging community experience.

8. Risks and Mitigation

The achievement and leaderboard system introduces technical, operational, and user-related risks, along with mitigation strategies leveraging ScummVM's existing architecture to ensure robust implementation.

8.1 Potential Risks

- *Security*: Risks include players manipulating local data, bots inflating leaderboards, and data breaches compromising sensitive information.
- *Performance*: Syncing achievement data may cause latency, disrupt gameplay, or overload servers during peak traffic.
- *Maintainability*: Increased code complexity and reliance on external services could complicate debugging and updates.
- *Usability*: Poor UI design or lack of accessibility may deter user adoption.
- *Privacy*: Sensitive user data, such as usernames or play history, may be exposed through public leaderboards.
- *Scalability*: A growing user base may overwhelm the system, affecting performance and data management.

8.2 Mitigation Strategies

- *Security*: Use server-side validation, encryption, CAPTCHAs, and secure authentication (e.g., OAuth 2.0). Conduct regular audits.
- *Performance*: Employ asynchronous syncing, caching, cloud-based scaling, and load balancing.
- *Maintainability*: Design the system modularly, document thoroughly, and use reliable hosting platforms.
- *Usability*: Conduct user testing to develop a simple, appealing UI with opt-in features and customization options.
- *Privacy*: Allow user preferences for public leaderboards and anonymous identifiers.
- *Scalability*: Optimize database performance through indexing, partitioning, and sharding.

8.3 Architectural Support for Risk Mitigation

ScummVM's architecture provides inherent mechanisms that support risk mitigation:

- The event-driven design ensures that new features integrate without interrupting the core gameplay loop, reducing the risk of disruptions.

- The Common subsystem offers centralized data handling, simplifying the implementation of redundant storage and error correction strategies.
- The decentralized architecture of ScummVM allows changes to specific components (e.g., Engines or GUI) without cascading impacts across the system.

By leveraging these architectural strengths, the enhancement can be implemented in a way that minimizes risks while maintaining ScummVM's reliability and performance.

9. Testing and Validation

To ensure the achievement and leaderboard system meets both functional and non-functional requirements (NFRs), a robust testing and validation plan will be implemented. This plan focuses on verifying core functionality, ensuring seamless integration with ScummVM's existing architecture, and addressing potential risks identified earlier.

9.1 Functional Testing Plans

The functional testing phase will validate that the achievement and leaderboard system performs its intended functions accurately. Achievement tracking will be tested to confirm that the Achievement Tracker properly logs game completion events and milestone achievements across all supported engines, such as the SCI Engine. Special attention will be given to edge cases, like partially completed games or skipped milestones, to ensure robust handling.

The leaderboard system will be tested for real-time updates, ensuring rankings reflect achievement events accurately. Filters for regional rankings or game-specific leaderboards will also be evaluated for correctness. Additionally, the notification system will be tested to confirm that players receive real-time updates when achievements are unlocked, even when multiple notifications are queued simultaneously.

9.2 Integration Testing with Existing Features

Integration testing will verify that the new components interact seamlessly with ScummVM's existing systems. In the Engines subsystem, the Achievement Tracker will be tested to ensure it integrates smoothly, with event listeners functioning without introducing performance issues or disrupting gameplay.

For the Common subsystem, integration testing will focus on validating that achievement data can be logged, stored, and retrieved effectively, without impacting existing functionalities like save states or configuration handling. The GUI subsystem will also be extended and tested to ensure the Achievements Screen and Leaderboard View are consistent with ScummVM's themes and provide a user-friendly experience. Usability testing will confirm that the interface enhancements meet player expectations.

The Backend subsystem will handle the storage and retrieval of leaderboard data. Integration tests will simulate frequent updates and large datasets to ensure efficient and responsive operations, supported by caching mechanisms to reduce latency.

9.3 Validation of NFR Compliance

Performance validation will involve stress tests to measure response times for achievement processing and leaderboard updates during heavy gameplay activity. Profiling tools will be used to identify and resolve bottlenecks in the Achievement Tracker and Leaderboard Management System.

To validate usability, user feedback sessions will evaluate the intuitiveness and functionality of the Achievements Screen and Leaderboard View. Notifications will be assessed for clarity and responsiveness during gameplay.

Scalability tests will simulate high user loads and extensive achievement data to confirm the system's ability to scale effectively. The caching mechanism will be evaluated for its role in maintaining leaderboard responsiveness under heavy usage. Reliability testing will ensure that data persists across sessions without loss or corruption. Additionally, scenarios like unexpected crashes will be introduced to verify that the system can recover gracefully.

10. Conclusion

The proposed achievement and leaderboard system introduces a modern, engaging layer to ScummVM, rewarding players for their accomplishments and fostering a sense of community. This enhancement integrates seamlessly with ScummVM's event-driven architecture, leveraging existing subsystems while respecting its core mission of retro game preservation.

By increasing user engagement through achievements, dynamic leaderboards, and intuitive interfaces, the system rewards classic gaming nostalgia and contemporary expectations. Future possibilities, such as online leaderboards or additional achievement types, provide avenues for further development and community growth.

This enhancement not only modernizes ScummVM but also strengthens its role as a platform for preserving and celebrating classic games, ensuring continued relevance and enjoyment for its users.

11. References

11.1 Cited Documents and Research Sources

1. ScummVM Team. (2023). *ScummVM main repository*. GitHub. Retrieved from https://github.com/scummvm/scummvm
2. ScummVM Forums. (2010). *How is ScummVM's Architecture Structured?* https://forums.scummvm.org/viewtopic.php?t=7886. Accessed 12 Nov. 2024.
3. SDL Wiki. *SDL3 FrontPage*. *n.d.*, https://wiki.libsdl.org/SDL3/FrontPage. Accessed 14 Nov. 2024.