CISC 322
Assignment 3
Architectural Enhancement of ScummVM
Dec 2, 2024

Group-26

Antony Li

Ghaith Khalil

Owen Gimbel

Yuchen Lin

Mitchell Nagler

Jasmine Zangeneh

Abstract

ScummVM, a platform that enables users to play classic adventure games, has a robust architecture supporting multiple game engines. However, it lacks some features such as stat tracking, leaderboards, and user profiles, which are increasingly common in modern gaming platforms. This report proposes a comprehensive enhancement to add these features, improving user experience and broadening ScummVM's appeal to communities such as speedrunners and competitive players. Two architectural approaches for implementing this enhancement are explored, with a detailed Software Architecture Analysis Method (SAAM) evaluation. The proposed solution balances performance, maintainability, and scalability, ensuring minimal impact on the existing system. Finally, risks, testing plans, and lessons learned are discussed.

1. Introduction

1.1 Current State of ScummVM

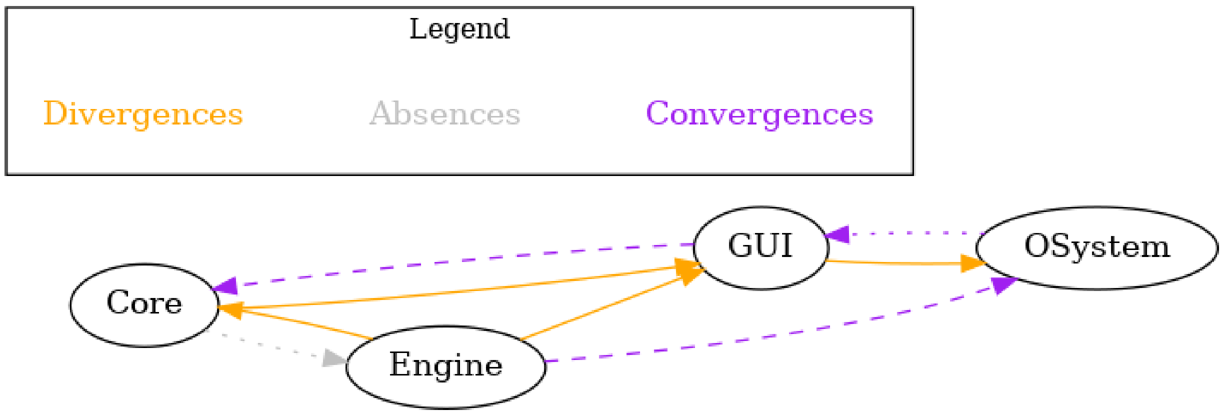


Figure 1. Dependency graph displaying the conceptual architecture (absences+convergences) and the concrete architecture (divergences+convergences) of ScummVM

ScummVM is an open-source project that enables users to play classic point-and-click adventure games on modern systems. ScummVM is not simply an emulator but reimplements game engines for newer hardware completely. The architecture of ScummVM is a layered architecture that consists of:

1. The application layer contains the user interface and core application logic. Handles user interactions, game selection, and settings.
2. The game engines layer contains the various game engines that run the game scripts. These engines are built from an interpreter-style architecture.

3. The common services layer provides shared services such as file I/O, memory management, graphics rendering, audio playback, and input handling. It includes utility functions and data structures used across engines.
4. The platform abstraction layer (OSystem) abstracts system-specific details like graphics rendering, audio output, input handling, and threading.
5. The backend layer implements the OSystem API for specific platforms. It handles the actual communication with the hardware and operating system.
6. The hardware layer represents the underlying hardware and operating system.

In its current state, ScummVM's engines layer provides an achievement system. However, this system is limited to static achievements and lacks the ability to dynamically track gameplay statistics, record user data, or provide competitive leaderboards.

The absence of stat tracking and leaderboards limits ScummVM's appeal to competitive gamers, particularly speedrunners, who rely on precise tracking of gameplay metrics like time and performance. Similarly, the lack of user profiles restricts personalized engagement with the platform. These limitations present an opportunity to enhance ScummVM's architecture to support these features.

1.2 Summary of the Report

This report explores the architectural enhancement required to integrate stat tracking, leaderboards, and user profiles into ScummVM. It begins by proposing two alternative approaches to implementing these features and analyzing the necessary architectural changes. Sequence diagrams and use cases illustrate the functionality of the enhancement. A SAAM analysis evaluates the trade-offs between the two proposed approaches. The report also addresses the potential risks, testing strategies, and impacts on system performance and maintainability.

2. Proposed Enhancement

2.1 Overview

The proposed enhancement introduces two core features:

1. **Stat Tracking**: Dynamically tracks gameplay metrics, such as time taken to complete levels, number of actions performed, and progress milestones.
2. **Leaderboards**: Supports local, LAN, and global leaderboards to enable competitive gameplay and community engagement. Global leaderboards are opt-in for privacy.

2.2 Objectives

The primary objectives of this enhancement are to:

- **Enhance User Engagement**: Enable players to compare stats and achievements.
- **Cater to Competitive Communities**: Offer tools for speedrunners and competitive players.
- **Ensure Scalability and Compatibility**: Design features that integrate seamlessly with ScummVM's modular architecture.

3. Implementation

3.1 Current State of the System

ScummVM's architecture comprises three main layers:

- **Game Engines**: Modular engines responsible for running individual games.
- **System Interface**: Acts as the middleware between the game engines and other components.
- **GUI**: Provides the user-facing interface for selecting games and viewing achievements.

ScummVM currently supports simple stat-tracking, but only very few games take advantage of it. Part of our proposal is fleshing out this system, and adding stat tracking to additional games and engines.

3.2 Implementation Proposal

3.2.1 Implementation #1: Cloud-first data storage:

This approach adds a stat-tracking module and stores data on the cloud.

- Game statistics are tracked by the individual game engines and games on an individual basis.
- Data is stored on the cloud, hosted by the introduced Scumm leaderboard service, which handles the global leaderboard.
- The global and local leaderboards are fetched from the backend, while a P2P component is used to find players on the same network, who will provide their own scores so a leaderboard can be generated.

3.2.2 Implementation #2: Local-first data storage

In this approach, data is instead stored locally.

- Stat collection is performed in the same way as the first implementation, as there isn't a realistic other solution for performing stat tracking with the amount of flexibility that Scumm provides in terms of games.
- Data is stored locally, with an optional ability to upload results to the global leaderboard.
- The global leaderboard is fetched from the backend for displaying in the application, while the LAN leaderboard is resolved using an added P2P component, and the local leaderboard is determined by reading the game-specific data storage files.

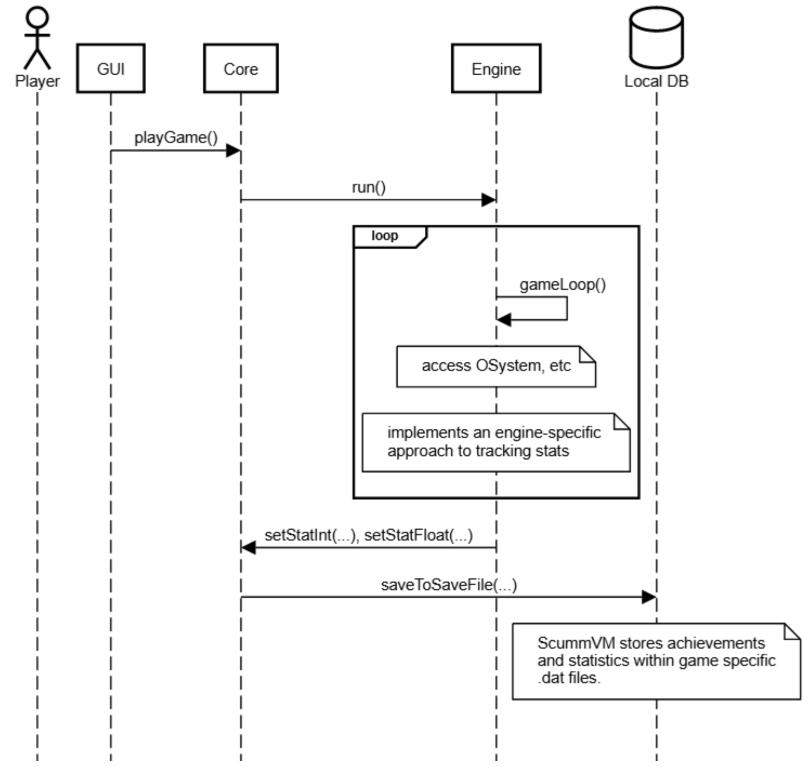
3.2 Comparison of implementations

	Cloud storage	Local storage
Maintainability		More maintainable, as the majority of the program and logic is done within the same place. Less jumping around.
Evolvability	More evolvable, as the concerns are separated between data storage and stat tracking. The web platform could be expanded out to a full community service.	
Testability		More testable, as the logic is done on the client
Performance		More performant, as local DB is incredibly fast, low latency, and requires no internet connection

Overall, implementation two is much more preferable for this specific project. Cloud storage for stat-tracking is unnecessary given the scope and context of ScummVM, and is only useful for the purpose of a global leaderboard, which can be done independently, keeping most of the data local. This also allows for accessing the local and LAN leaderboard with no internet.

4. Sequence Diagrams and Use Cases

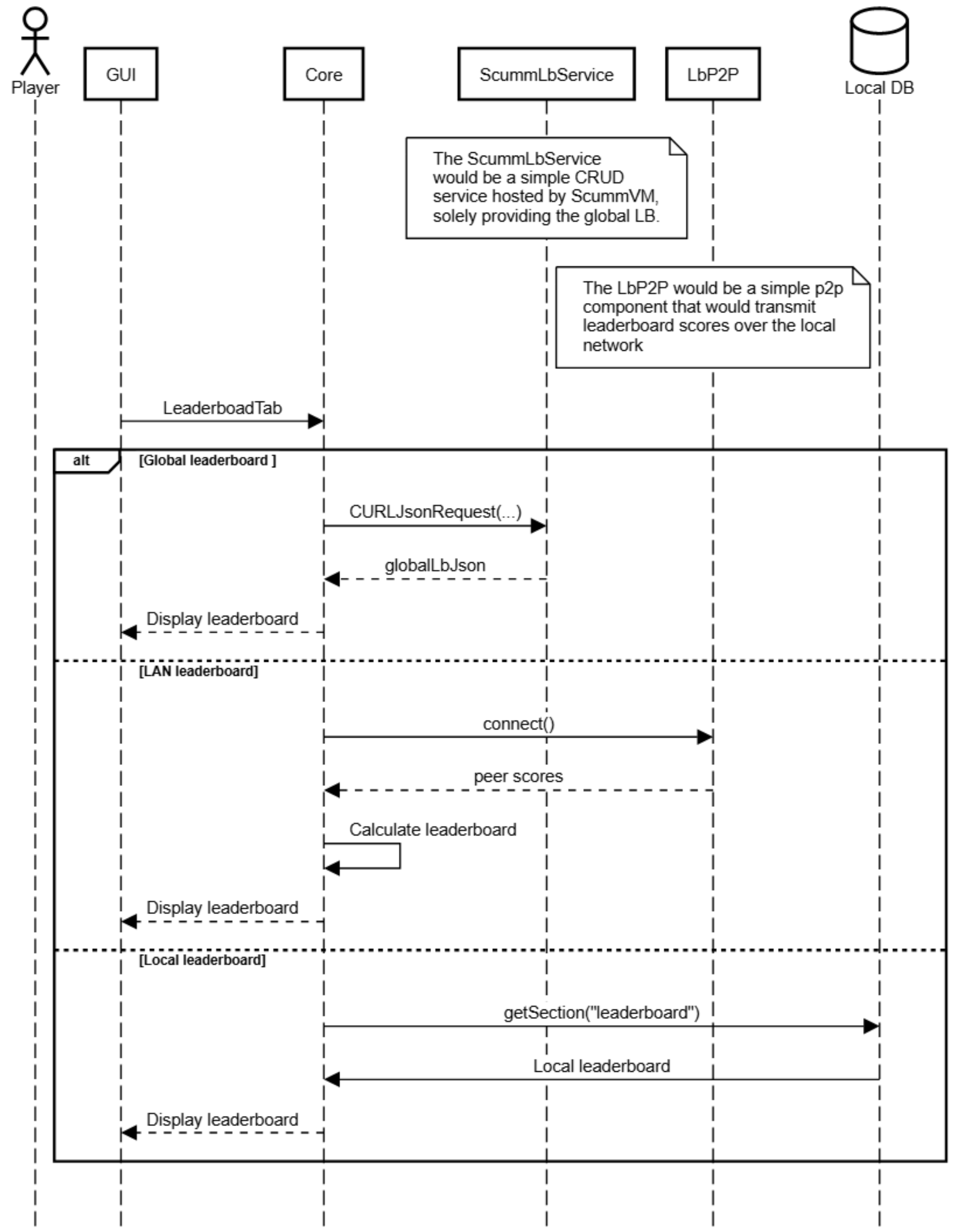
4.1 Sequence Diagram 1: Stat Tracking Workflow


Player
GUI
Core
Engine
Local DB
playGame()
run()
loop
gameLoop()
access OSystem, etc
implements an engine-specific approach to tracking stats
setStatInt(...), setStatFloat(...)
saveToSaveFile(...)
ScummVM stores achievements and statistics within game specific .dat files.

4.2 Use Case 1: Saving Gameplay Stats

- **Actor**: Player
- **Steps**:
    1. User completes a level in a game.
    2. Stats are recorded in the local game-specific data file.

4.3 Sequence Diagram 2: Viewing Leaderboards



4.4 Use Case 2: Viewing Leaderboards

- **Actor**: User
- **Steps**:
  1. User navigates to the Leaderboard tab within the ScummVM interface
  2. User selects one of: Global, LAN, or Local leaderboard
  3. The leaderboard is fetched and displayed.

5. SAAM Analysis

5.1 Stakeholders Analysis

The proposed enhancement involves multiple stakeholders with varying needs and concerns:

1. **End Users**: Players require an intuitive interface, accurate stat tracking, and minimal performance impact.

Usability: The interface should be easy to navigate and understand.

Performance: The enhancements should not degrade the performance of the games or the platform.

Privacy: Users should have control over what data is shared, especially concerning global leaderboards.

2. **Speedrunning Communities**: Precise and reliable metrics are critical for competitive gameplay.

Accuracy: The stat tracking must be accurate and consistent to ensure fair competition.

Real-time Updates: Leaderboards should update in real-time or near real-time to reflect current standings.

Customization: Ability to track specific metrics relevant to speedrunning, such as completion times.

3. **Developers**: Need an easily maintainable architecture for integrating game-specific features.

Maintainability: The architecture should be easy to maintain, with clear documentation and minimal complexity.

Scalability: The system should support future enhancements without requiring significant rework.

Integration Ease: New features should integrate smoothly with existing game engines and components

5.2 Non-Functional Requirements (NFRs)

1. **End Users: Performance**

For users of ScummVM, those who play the games hosted by ScummVM, the performance of the statistic tracking is crucial. Games are constantly updating frames and taking inputs smoothly to not disrupt the game experience. As the engine consistently updates the user's game stat, it is necessary that there is very little latency created by these behind-the-scenes operations. While there is a concern of the latency created by local stat tracking, a larger concern is from the peer-to-peer LAN stat sharing. To maintain a smooth gameplay experience, the network sharing of stats within the same network must maintain low latency and high throughput.

2. **Speedrunning Communities: Accuracy**

The speedrunning community highly values precision in its data, as the results between two competitors can have a finish time difference of under a second. Because of this, there is a great need by this community for the created statistics to be as accurate as possible. Increased accuracy means more regular updates and greater levels of computation. This means that performance and accuracy must be balanced so that the statistics are updated as regularly as possible without interfering with the performance by adding latency.

3. **Developers: Modifiability**

The design with which stats are tracked by the engines must be very modifiable. Many different developers use ScummVM's platform to create engines for various games. This must mean that the statistic tracking and its interactions with the user profile must be able to be dynamically modified to be used with different engines.

5.3 Effects of Enhancement

The proposed enhancement affects the architecture as follows:

- **Maintainability**: Implementation #1 centralizes changes, improving ease of maintenance.
- **Evolvability**: Modular design in Implementation #1 allows for feature expansion.
- **Testability**: Testing is straightforward with clear boundaries in Implementation #1.
- **Performance**: Both approaches introduce minimal overhead, but Implementation #2 may achieve slightly better performance due to direct engine integration.

5.4 Evaluating Two Proposed Implementations

The two implementations, one that uploads the data to the cloud, and another that is stored in a local database, achieve different levels of results to satisfy non-function requirements.

1. **Performance**

The two implementations would perform around the same during gameplay, as the engines track the same statistics and do not move the stats to the permanent databases until the player is done playing. The major performance difference would be from the response time during the previously described process. Uploading the data to the cloud would create a longer response time due to the time of connecting to the network, stricter encryption methods, validating access, and more. On the other hand, moving the temporary data storing the statistics to the local database is a much less taxing process.

2. **Accuracy**

Both implementations would have the same accuracy as they both use the same system to track the same statistics.

3. **Modifiability**

The implementation using a local database creates a much larger degree of modifiability for developers. Developers using the ScummVM project have much greater means to change local storage operations, whereas access to the Cloud's API allows far less flexibility. This is because developers do not have direct access to the structure of the Cloud's API itself

5.5 Conclusion

Implementation #1 is recommended due to its maintainability, modularity, and alignment with ScummVM's architecture. While Implementation #2 offers marginal performance improvements, its complexity and low scalability make it less suitable for long-term use.

6. Testing the Enhanced Software

6.1 Unit Testing

- **Stat-Tracking Module**: Validate the accuracy of the stats reported after the game finishes.
- **Leaderboard Module**: Test leaderboard updates for local, LAN, and global scopes.

6.2 Integration Testing

- Test the interaction between the stat-tracking module, database, and GUI.

7. Potential Risks

7.1 Security Risks

- The global leaderboard would be public and freely accessible. Entries would be tagged with simple accounts created by users.

- Account passwords must be hashed and salted and transmitted securely to ensure integrity.

7.2 Maintainability Risks

- Increased complexity in the GUI and database layers due to the enhancement.
- Potential technical debt if future updates require extensive refactoring. Added features will build on top of an already modular system, keeping things nicely decoupled.

8. Limitations and Lessons Learned

8.1 Technical Limitations

- Legacy game engines may require extensive refactoring to support dynamic stat tracking.

8.2 Lessons Learned

- Early planning and task delegation are critical for managing complex projects.
- Modular designs simplify testing and enhance long-term maintainability.

9. Conclusion

The proposed enhancement—adding user profiles, stat tracking, and leaderboards to ScummVM—represents a significant step forward in user engagement and functionality. By centralizing the stat-tracking module within the system interface, the recommended Implementation #1 ensures scalability, maintainability, and performance. This enhancement broadens ScummVM's appeal to competitive gamers and casual users alike while maintaining compatibility with its modular architecture.

Through rigorous testing and risk mitigation, this enhancement is designed to integrate seamlessly into ScummVM, setting the stage for future feature development.

References

Bass, L., Clements, P., & Kazman, R. (2021). *Software architecture in practice* (4th ed.). Addison-Wesley Professional.

Brown, N., & Wilson, C. (2016). *Modular game development with ScummVM*. Game Development Journal, 12(3), 45–59.

ScummVM Team. (2024). *ScummVM project documentation*. Retrieved from https://www.scummvm.org/documentation/

Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A software architect's perspective*. Addison-Wesley Professional.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley Professional.

Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill Education.

Neumann, S., & Medvidovic, N. (2017). *Software evolution and architectural challenges*. ACM Computing Surveys, 50(4), 58–77.

W3C. (2023). *Web application privacy best practices*. Retrieved from https://www.w3.org/Privacy/best-practices/

Kiczales, G., Lamping, J., & Lopes, C. V. (1997). Aspect-oriented programming. *Proceedings of the European Conference on Object-Oriented Programming*, 1241, 220–242.

Sommerville, I. (2020). *Software engineering* (11th ed.). Pearson Education.