1000101: ScummVM Enhancement Proposal

Jasper Nie (20jhn3@queensu.ca)

Vlad Lisitsyn (vlad.lisitsyn@queensu.ca)

Alex Ivanov (22ai11@queensu.ca)

Isaac Chun (21ihc1@queensu.ca)

Benjamin Bartman (20bb27@queensu.ca)

Sabin Pokhrel (sabin.pokhrel@queensu.ca)

CISC 322

Software Architecture Conceptual Report

December 2, 2024

Contents

1 Abstract

This report proposes adding cloud save support to ScummVM to enhance its usability and ensure it meets the expectations of modern gamers. Cloud save functionality will allow users to continue their game progress across multiple devices, safeguard save files against hardware failures, and eliminate the need for manual file transfers. This enhancement will directly improve ScummVM's usability, adaptability, and long-term relevance.

The report begins by reviewing ScummVM's current save system and its limitations. It then outlines the proposed enhancement, detailing its interactions with existing features and the necessary changes to the architecture. Two implementation strategies are evaluated: developing a dedicated ScummVM cloud service or leveraging existing cloud platforms. Using a SAAM analysis, the report demonstrates why the ScummVM-specific approach is the best option for scalability, maintainability, and user experience. Finally, the report highlights the effects of this enhancement on the system's performance, maintainability, and user experience, concluding with the changes required to support this new functionality.

2 Introduction and Overview

ScummVM has long been a reliable platform for running classic graphical adventure games on modern devices. Its modular architecture ensures compatibility with various platforms and provides a solid foundation for further improvements. However, one area where ScummVM lags behind modern gaming systems is its save functionality, which is restricted to local storage. This limitation forces users to manually transfer save files between devices, creating friction and inconvenience.

This report introduces cloud save support, an enhancement designed to address these limitations. Cloud save functionality will allow users to save their progress online, ensuring that game states can be accessed across multiple devices. This feature aligns with the growing demand for cross-device compatibility and data security in gaming. It also provides ScummVM with an opportunity to modernize its offering and increase its appeal to a broader audience.

The report explores the current state of ScummVM's save system, the proposed cloud save enhancement, and its interactions with existing subsystems. It also evaluates alternative implementation approaches, analyzing their strengths and weaknesses through a SAAM framework. Finally, the report outlines the architectural changes required to support cloud saves and the testing strategies to ensure a smooth integration.

3 Proposed Enhancement

Adding cloud save support to ScummVM is a great addition to the system and increases its viability in the long term. The reasons for this are its usefulness in the user experience and the many improvements it brings to system functionality

Having cloud save support be a part of ScummVM allows for cross-device continuity. Many ScummVM users access the platform on multiple devices such as laptops, desktops and mobile devices. Having this functionality allows players to seamlessly continue their gameplay from wherever they left off, regardless of the device being used. Furthermore, cloud saving acts as a backup that ensures no data is lost to local hardware failures or accidental deletion of files on your device. It also saves the hassle of having to manually transfer files between devices when wanting to switch over to a different device while keeping the same save file. This feature is an expectation in modern gaming platforms that ScummVM should have if it wants to adhere to the standard that gamers are used to with their day to day gaming experience.

In terms of system level improvements, allowing users to pick up their game from any device encourages longer and more consistent gaming sessions, which can boost user retention.

As previously mentioned, this is also a feature that is not only expected but required if ScummVM wants to keep looking forward into the future and finding ways to stay relevant in the increasingly connected gaming world. Cloud saving also opens avenues for integration with popular platforms like Google Drive, Dropbox, or any game specific ecosystems which can further increase the software's ecosystem compatibility. Centralizing save data like this in the cloud simplifies the testing of games across platforms for both the developers and the contributors, reducing friction when sharing and debugging save-related issues or any associated problems with the system.

4 Interactions with Other Features

4.1 Save/Load (Conceptual) / Common (Concrete)

Cloud save support directly extends this functionality. Instead of having to rely solely on local file storage, cloud integration adds an additional layer for handling save data. This enhances the modularity and scalability of the system. The way these would interact is the local save files will be synchronized with the cloud which will require the subsystem to support both storage areas. The Common module identified in the concrete architecture will need to contain the necessary API's for cloud storage providers, which will handle the upload and download processes.

4.2 Platform Abstraction Layer (Conceptual) / Backends (Concrete)

Adding this feature means that the platform abstraction layer will need to abstract differences in the different cloud service APIs that are implemented (Google Drive, Dropbox) to be sure that independent platform implementation is done for each. Furthermore, backend configurations are going to need to address device specific storage constraints such as offline modes and cache management. This integration takes advantage of the Backends module design to maintain cross platform compatibility, ensuring consistent cloud save functionality across all supported devices.

4.3 File I/O Handler (Conceptual) / Common, Base (Concrete)

New methods must be added to this component to be able to upload and download save data from the cloud. Additionally, because of cloud save being added there could be conflicts between local and cloud saves that will need to be addressed through conflict resolution mechanisms in this module. Even though this introduces some additional complexity to the Common module, centralizing these interactions aligns better with the modular design of ScummVM which will make debugging and updating the system easier in the future.

4.4 User Interface (Conceptual) / GUI (Concrete)

In this module, elements such as Save to Cloud, Load to Cloud, and syncing indicators will need to be added. There will also need to be added progress indicators for upload and downloads as well as error messages associated with synchronization of data throughout cloud save data on multiple devices. This will improve the overall user experience while requiring very minimal additions to be made to the GUI.

4.5 Debugging and Error Handling (Conceptual) / Common (Concrete)

In order to patch the new points of failure that will be added when cloud saving is introduced, new error codes and logging mechanisms will need to be added for cloud related failures. There will also need to be error recovery mechanisms such as retrying failed uploads which need to be implemented into the system. These changes will improve the robustness of the

system but will require careful design to ensure that errors are not disrupting gameplay or corrupting save files.

4.6 Game Database Engine (Conceptual) / Common (Concrete)

The Common module, which stores game specific metadata and configurations will need to be synchronized with the cloud. This will ensure that metadata such as game version, save file compatibility and timestamps will all be properly updated within the cloud. This will also mean that the system will be able to automatically detect cloud saves for specific games based on the database, which enhances the system's ability to manage save files across multiple devices.

5 Current State of the System

ScummVM currently relies on a local save system, where game progress is stored directly on the user's device. The Common module manages the creation and storage of save files, while the File I/O Handler ensures efficient file operations. Although these components are effective for local file management, they lack the flexibility needed for cloud integration.

The Platform Abstraction Layer allows ScummVM to work across many devices, but it isn't designed to handle differences in cloud storage services or to synchronize data between platforms. Similarly, the Debugging and Error Handling tools are focused on local file issues and don't account for problems that could happen with cloud integration, like upload failures or file conflicts.

While this architecture has served ScummVM well for local operations, it cannot meet the demands of modern gaming, where seamless cross-device functionality is often expected. This gap highlights the need for cloud save support to extend ScummVM's capabilities.

6 Effects of the Enhancement

6.1 Evolvability

Introducing cloud save functionality prepares ScummVM for future enhancements and changing user needs. The modular nature of the proposed design allows for the addition of new cloud providers or features, such as offline save caching, without requiring significant changes to the rest of the system. This enhancement also provides a scalable foundation that can grow alongside ScummVM, ensuring it remains relevant as user expectations and technology evolve.

6.2 Maintainability

The centralized management of save files through a cloud-based system simplifies debugging and reduces the effort required to update or modify the save functionality. By organizing cloud interactions within the Common and Backends modules, the system remains modular and easier to maintain. Any issues related to synchronization or storage would be isolated within these modules, reducing the risk of unintended side effects when making changes.

6.3 Performance

While cloud synchronization might introduce minor delays, the impact on system performance is expected to be minimal. Efficient use of caching and asynchronous operations in the Backends module can mitigate potential slowdowns. Additionally, since cloud saves are an optional feature, users who rely solely on local saves will experience no performance degradation. The modular design ensures that cloud functionality operates independently, minimizing its effect on core gameplay processes.

6.4 Testability

Cloud save support makes testing easier and more thorough. Developers can validate save functionality across different devices and platforms by replicating user scenarios. For instance, synchronization processes and conflict resolution mechanisms can be tested in controlled environments to ensure data integrity. The feature also simplifies debugging by centralizing save data, making it easier to isolate and reproduce issues.

7 Alternatives

We considered two main options for adding cloud save support:

7.1 Building a ScummVM Cloud Service

This option involves creating a custom cloud system specifically for ScummVM. The architectural style for this solution would follow a client-server architecture, where ScummVM acts as the client and interacts with a centralized server to manage save data. This approach allows for tight integration with ScummVM's existing modules, providing full control over features like synchronization, scalability, and security. By centralizing data storage, this style also simplifies the implementation of cross-device compatibility and conflict resolution.

However, the client-server model requires significant development effort to build and maintain the server infrastructure. Additionally, ensuring robust security and reliability for the server would require ongoing management, which could add to the workload.

7.2 Using Existing Cloud Services

This option leverages external platforms like Google Drive or iCloud. The architectural style here is more of a layered architecture, where ScummVM interacts with cloud storage providers through APIs provided by these services. The Backends module would act as a mediator, abstracting the specifics of each platform to maintain compatibility across multiple providers.

While this layered approach reduces the burden on ScummVM developers by offloading storage responsibilities to established providers, it also introduces dependencies. Each cloud service API has its own design and constraints, making maintenance more complex and potentially leading to inconsistent user experiences across platforms.

Why ScummVM cloud service is recommended: After reviewing these alternatives, we recommend the ScummVM Cloud Service. The client-server architecture aligns better with ScummVM's modular design and offers the scalability and flexibility needed for long-term growth. It allows ScummVM to maintain control over user data and adapt the system to meet future requirements without being tied to third-party platforms.

8 SAAM Analysis

The two ways to realize the enhancement of cloud storage for user's game save state are creating ScummVM's own cloud storage service and using platform-specific cloud storage services (Steam for PC, iCloud for IOS, Google Play for Android). The major stakeholders impacted with this enhancement would be users and the software engineers of ScummVM, as users will now have the ability to save their data to an account and access it across multiple platforms and the engineers have to build the architecture and software to make it happen.

Important NFRs for the user stakeholder include performance, portability, availability, and security. The user will want to be able to access their saved data in a fast amount of time, will want to be able to access their data files across multiple platforms, will want to

be able to access their files at any time, and will want their information and data to be secured and encrypted. Important NFRs for the engineer stakeholder include reusability, maintainability, capacity, manageability, and security. The engineer will want to be able to make their code into a component that can be used across other softwares, will want to easily maintain the code over a long period of time plus quickly make changes and fixes to the code, and securely store user data and protect them from attacks.

Focusing on the ScummVM cloud service option, this implementation affects the NFRs of the user stakeholder as the performance of the storage would be consistent over all unique platforms and would be dependent on the backend choice of the developers, the portability of the storage would be high as users would just have to have their data converted into the proper file type for differing platforms, the availability of the storage would still be dependent on the backend chosen by the developers and how robust the software is, and the security of data would be more flexible under a ScummVM specific storage, but again would rely completely on the engineers to develop a safe storage. This implementation affects the NFRs of the engineer stakeholder as the reusability of the code is very high as the functionality lies within one central structure, the maintainability of the code is centralized to one component that the engineers must be technically sound with interacting with, the capacity of the storage being easily scalable, and the security of the storage requiring the engineers to be experienced with technologies, similar to how users view security.

Focusing on the platform-dependent cloud service option, this implementation affects the NFRs of the user stakeholder as the performance will be dependent on the platform (Steam, iCloud) but overall will be higher due to how robust those platforms already are, the portability is low as locking users to an existing platform will reduce flexibility of users, the availability of the storage would be high as these platforms are linked to much bigger and more reliable companies (Google, Apple), and security would be very robust due to being linked to these reliable companies as well but still being less flexible. This implementation affects the NFRs of the engineer stakeholder as the reusability of the code is very low due to each platform requiring their own codebase and implementation, the maintainability of the the code is arduous due to having to manage the storage of multiple platform softwares, the capacity of the storage being offloaded to the third-parties which requires no additional infrastructure, and the security of the data being handled by the third-parties as well.

Considering the previous analysis, the preferred implementation is going to be based on the scale that ScummVM operates. The platform-dependent cloud service option provides very limited scalability for the engineers, which could result in worse performance and security for users if the capacity is too much. However, this implementation would be much easier to get working in comparison to a ScummVM cloud service, which would require much more work and skill needed. The ScummVM cloud service carries the benefits of easy scalability and reusability of code, which would be preferred for a larger user base. Noting this, we believe that the ScummVM cloud service would be the best option, as although it would take more work to initially create the functionality, the easy scalability and customization will save much more time and effort when the user base increases versus the strict limits of platform-dependent cloud services.

9 Impacted Directories

9.1 Common Module

This module will need APIs to interact with cloud storage providers, handling operations such as upload, download, and synchronization of save files.

9.2 Base Module

Temporary storage of save files will be managed here during cloud synchronization processes.

9.3 Backends

Updates will ensure compatibility with various cloud service APIs and address platform-specific constraints like offline support and caching.

9.4 GUI Module

New interface elements, such as progress bars, error messages, and synchronization options, will be added to enhance user interaction.

10 Plans for Testing Impact of Interactions

The impact of the interactions between the cloud save system and existing features can be tested by benchmarking ScummVM in a number of scenarios. The first scenario is the base case of a user not using the cloud save feature. If the user is not using the feature the performance of ScummVM should be identical to how it was before the features implementation. If ScummVM is slower that means that there are some new dependencies that are performing extra operations that they should not and the code should be revisited to make sure the cloud save code is only used when needed. The second scenario is testing how the cloud save system stores and updates existing files. It is important that the system does not bloat file sizes or create unnecessary files, ideally it should simply replace existing saves with the cloud saves if the cloud save is newer than the existing save. If the files turn out to be larger than they were beforehand or if the system is not properly uploading or downloading cloud saves then the code relating to file I/O should be examined. The third scenario is testing edge cases like having no internet, or no saves or having corrupted saves in the cloud or locally. The system should be able to detect if the saves are valid and if the computer has a connection so as not to store useless saves or fail to store saves. The final scenario is testing the general system load of the cloud save process. As the process occurs the computer's CPU, RAM, and network usage should be closely monitored to ensure that there are no unnecessary spikes in usage and that the process occurs with minimal drain on resources. Cloud saving is not a resource intensive process so if it turns out that a lot of resources are being used the code should be examined further to ensure that there are no hanging loops or other methods that hog resources.

11 Potential Risks

A cloud save system should not have many risks as it is a relatively simple system, however there are still some things that should be considered. First is security, it is important to make sure that the system validates save files before downloading them from the cloud. If someone were to hijack the user's cloud and upload some malicious file instead of their save, ScummVM should be able to detect that something is wrong and not download the file. Another factor to consider is system support. ScummVM is designed to run on a large variety of systems and these systems often vary in how they store files and how they can connect to the internet. The cloud save system should communicate with the backends subsystem to understand what kind of device it is running on and how to perform the needed operations to store and upload saves. Maintainability is also an important factor. ScummVM should not exclusively rely on a determined third party service to store saves as if the service were to encounter an issue it could compromise the usability of the cloud save feature. A solution to this problem could be allowing the user to choose from a list of services such as Google Drive or OneDrive, so that the user has some agency in regards to what service is being used to store their save. The final potential risk is loss of info through the upload/download process. If there was some error that occurred during the download or upload of the save file it could corrupt the file or otherwise modify it in some unwanted way. ScummVM should create backups of save files to

make sure that if some issue were to occur, the save files could be rolled back to a working version.

12 Diagrams

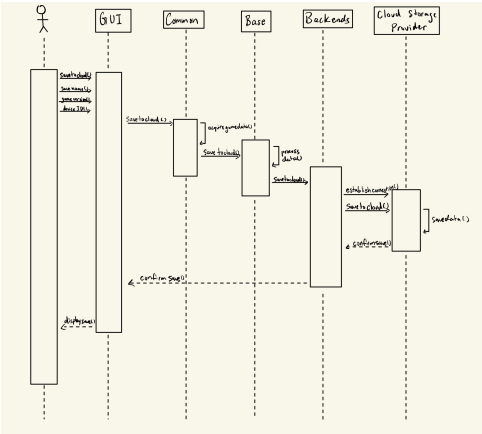

Figure 1: Use Case #1 - Saving a Game State to the Cloud

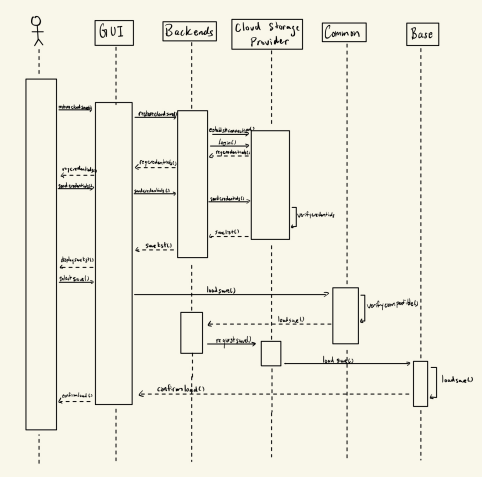

Figure 2: Use Case #2 - Restoring Game State from the Cloud to the Local Device

13 Use Cases

13.1 Saving a Game State to the Cloud

The user saves their game state to the cloud for later restoration.
Actors:

- User
- Cloud Storage Provider
- GUI
- Base
- Common
- Backends

Preconditions:

- The game is running, and the player is actively playing.
- The user has access to a valid cloud storage account.

Flow of Events:

1. The user selects "Save to Cloud" from the GUI.
2. The GUI prompts the user to provide metadata, such as the save name, game version, and device ID.

3. Common retrieves the current game data, player progress, and metadata, such as the game name, player settings, and save timestamp.
4. Base temporarily stores the game state and metadata in a local file.
5. Backends establishes a connection with the Cloud Storage Provider to upload the save data.
6. The Cloud Storage Provider accepts and stores the save data, associating it with metadata like save name, player, and timestamp.
7. The GUI shows a success message to the user after the save is successfully uploaded.

Post Conditions:

- The game state is successfully uploaded to the cloud and stored with associated metadata.

13.2 Restoring Game State from the Cloud to the Local Device

The user restores a previously saved game state from the cloud to continue their progress.
Actors:

- User
- GUI
- Backends
- Cloud Storage Provider
- Base
- Common

Preconditions:

- The game is running, and the player is logged into the cloud storage provider.
- There is an existing save available in the cloud storage.

Flow of Events:

1. The user selects "Restore Cloud Save" from the GUI.
2. The GUI prompts the user to log into the cloud storage provider if not already authenticated.
3. Backends sends a request to the Cloud Storage Provider to fetch a list of available saves associated with the user's account.
4. The Cloud Storage Provider responds with a list of saved game files, including metadata such as the device, game version, and save date.
5. The GUI presents the user with the list of available cloud saves, allowing them to select one to restore.
6. Common validates the selected save by checking the file's compatibility with the current game version and ensuring it matches the correct device ID.
7. Backends retrieves the selected save file from the Cloud Storage Provider.

8. The Cloud Storage Provider sends the save data to Backends, which downloads the file to the local device.
9. Base restores the downloaded save data to the local file system and prepares it for use in the game.
10. The GUI shows a success or error message to the user based on the outcome of the restoration process.

Post Conditions:

- The game state is successfully restored to the local device from the cloud, and the player can continue from where they left off.

14 Naming Conventions

UI: User Interface
Pub-Sub : Publish/Subscribe
GUI: Graphical User Interface
API: Application Programming Interface
I/O: Input/Output

15 Conclusion

In conclusion, this report was made to explain how cloud save support could be implemented into ScummVM's architecture, both on the concrete and conceptual side of things. The enhancement was explained, showing why it should be implemented into the system and how it could be done looking at the various interactions with other modules it would have. Increasing system viability for the future, user experience as well as system functionality are three key points of the system that are improved when adding this enhancement. Implementation options were discussed alongside their architectural styles, highlighting why creating a cloud service based on ScummVM would be the way to go. This was based on the increased flexibility that it offers as well as customization for scalability. The report also provided an overview of the many effects the enhancement has on both the high and low level conceptual architectures of the system, revealing potential changes that need to be made as well as identifying the directories that were impacted when introducing cloud save support into this system. After careful analysis, it is clear that enhancement of the system through this feature would be a task worth taking on because of the many upsides it brings. There are a few risks which have been identified in the report, but these can be reduced or mitigated enough to justify having cloud save support be a part of ScummVM.

16 Lessons Learned

One of the most important lessons we learned throughout this report was the importance of evaluating trade-offs when proposing a system enhancement like cloud save support. Every new feature of course comes with benefits but also different drawbacks, and identifying these drawbacks and making sure they do not outweigh the benefits is a key. In this scenario, we identified increased complexity as well as potentially being dependent on third party APIs to be key challenges faced when implementing this feature. Performance overhead was also something to consider when looking at how key factors such as network latency, data synchronization and conflict resolution could determine the success of this enhancement. To address these trade offs, we proposed redesigning the abstraction layer in a way that it could handle either its own cloud storage system or third party APIs. We also proposed adding comprehensive error handling mechanisms that will help solve any issues that arise when

dealing with the cloud. Balancing these trade offs showed us that architectural enhancements must be properly designed with any potential drawbacks held in consideration. Evaluating the various effects of a new feature on a system's maintainability, performance as well as user experience can help a system such as ScummVM succeed in the long run.

17 References

https://www.scummvm.org/
https://wiki.scummvm.org/index.php?title=Developer_Central