ScummVM Architectural Enhancement

Authors

Nick Axani	21nra@queensu.ca
Trajan Brown	21tjnb@queensu.ca
Aaron Chen	aaron.chen@queensu.ca
Gwendolyn Hagan	21gwh4@queensu.ca
Jack Hickey	jack.hickey@queensu.ca
James Kivenko	21jk90@queensu.ca

Table of Contents

Abstract

This report analyzes possible architectural changes in the ScummVM GUI system, analyzes architectural changes to decouple the Launcher GUI from the broader system for better modularity, scalability, and maintainability. The following discussion takes into account some drawbacks, risks, and benefits of two suggested approaches: a) implementing one system responsible for both launcher and in-game overlays, and b) implementing a separate Launcher GUI component which would contain sub-components to minimize code duplication. Whereas Method A gives a streamlined architecture and reduced duplication, it does so at higher implementation cost and risk. Method B minimizes structural changes for stability and simplicity but still enables moderate gains in code reuse and modularity.

Key stakeholders-ScummVM Coders, Game Players, and Engine Developers-are analyzed in terms of their prioritized non-functional requirements such as performance, portability, and reusability. A review of external interfaces that are impacted by this architectural change includes operating systems, file systems, and game engine GUIs. It highlights that very few disruptions will occur but allows for improvements to be made to the GUIs.

Ultimately, the balance of practicality and advancement that Method B strikes will achieve significant architectural improvements with reduced risk and effort compared to Method A. This approach will support ongoing development while keeping ScummVM committed to its principles of accessibility and cross-platform functionality.

Introduction

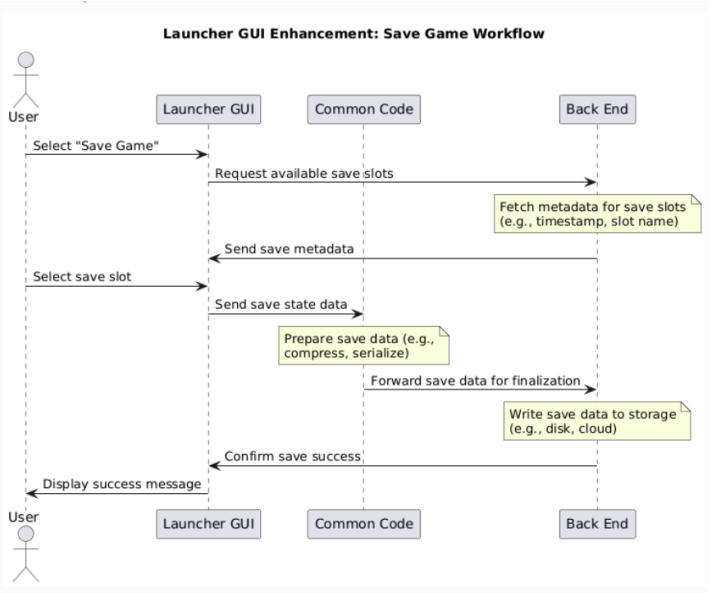
ScummVM is renowned for its modular architecture, which enables it to support a variety of classic game engines while maintaining cross-platform compatibility. Its design prioritizes adaptability and scalability by separating key responsibilities across subsystems. Despite ScummVM's overall versatility, certain components, such as the Graphical User Interface (GUI), remain tightly coupled with the Launcher Subsystem. This coupling introduces challenges that limit ScummVM's ability to scale and adapt to modern requirements, such as implementing multiplayer functionality or improving user accessibility.

This report focuses on an architectural enhancement to fully separate the in-game overlay GUI and the launcher GUI into distinct, independent subsystems. This decoupling will allow the GUI components to evolve independently, aligning ScummVM's concrete architecture more closely with its conceptual design goals. By isolating the GUI's responsibilities, the system will benefit from improved maintainability, enhanced scalability, and increased flexibility for future expansions. These changes also reduce interdependencies between the Launcher Subsystem and GUI functionalities, making testing, debugging, and feature additions more straightforward.

The proposed enhancement impacts several key subsystems:

- **Common Code:** Adjusting dependencies related to graphics, audio, and key mapping for clearer modularity.
- **Back End:** Redefining responsibilities for user input and save state handling to minimize coupling.
- **Engines:** Reorganizing in-game overlay components to improve separation of concerns and support modular design.

This report will analyze the proposed architectural enhancement in detail, evaluating its benefits, required modifications, and potential drawbacks. Additionally, the report includes a Software Architecture Analysis Method (SAAM) evaluation to compare two approaches for realizing the enhancement and identifying trade-offs and impacts on stakeholders. These alternatives will be assessed for their stakeholder impact, including developers, engine maintainers, and end-users, along with their alignment with non-functional requirements like modularity, performance, and scalability.



Sequence Diagram: Ideal save game workflow with proposed enhancements, redundant dependencies removed

Architectural Changes

Benefits

There are 3 primary benefits to separating the code that generates and manages the GUI of the ScummVM software into its own subsystem.

Compartmentalization

Currently, though much of the code corresponding to the GUI elements is contained in our Launcher Subsystem component, the graphical code relating to said GUI can also be found spread through both the Game Engines and Common Code layers as well. To simplify a system as large and complex as ScummVM, we want to be able to break it down into conceptually separate components to make it more palatable to our understanding. This is what we have done throughout the previous two assignments to nurture a more full understanding of ScummVM. Through the separation of the GUI-related files into their own subsystem within ScummVM, we achieve several things at once:

Firstly, this provides a clearer division of subsystems within ScummVM, which in and of itself is valuable to ScummVM developers.

Secondly, some of the code that we had previously placed in the Bottom Level layer of our concrete architecture will be transferred into the new Launcher GUI component. The transference of this code to a more niche and specified subsystem will further clear up the Bottom Level layer's purpose and function within the ScummVM. Notably, much of the code would come from the Launcher Subsystem component, clarifying its purpose as managing

the operations of the launcher for ScummVM rather than all of the currently existing GUI elements that are so deeply intertwined with it.

Thirdly, much of the refactored code will be from the Common Code layer. The transference of this code away from the Common Code layer further clarifies the role of the Common Code layer within our understood architecture of the ScummVM system. Our previous analysis of the Common Code layer included a lot of code which seemed to serve rather nebulous purposes within the ScummVM software. For example, the code that served for general processing, interfacing between the Game Engine subsystem and the Launcher Subsystem within the Bottom Level layer. Through the removal of GUI-related files and code from the Common Code layer, the Common Code's purpose is further refined into a layer that generally serves as a tool box for the Game Engine layer and Bottom Level layer.

Altogether, by further separating the GUI-related code and files into their own subsystem of ScummVM we not only clearly quarantine those files for further inspection and upkeep, we also further cement the purposes of other, previously defined components of our concrete architecture for ScummVM.

Resembles Conceptual Architecture

Another benefit of separating the GUI-related files and code into its own subsystem is that it more closely resembles our conceptual architecture for the system. Originally, when we had conceived of our conceptual architecture, we had placed a component for each of the Game GUI and Launcher GUI with a dependent relationship between only the Game Engines subsystem and Launcher Subsystem components respectively. During our creation of the concrete architecture, we found that the code relating to the GUI component that we'd previously considered in our conceptual architecture were actually spread across multiple files with multiple purposes through multiple components and so the decision was made to scrap the component for our concrete architecture. We had ultimately concluded that there was not enough of a basis for it within the actual organization and code of ScummVM to warrant calling the Launcher GUI its own subsystem.

With the prospect of adding additional code to ScummVM, it becomes feasible to separate the Launcher GUI from the other concrete subsystems to which it is so heavily tied. In doing so, we could create a version of ScummVM that more accurately reflects our initial intentions when we set out to define a conceptual architecture. While resembling our conceptual architecture is not necessarily a benefit on its own merits (we do expect our concrete architecture to differ from our conceptual), this is beneficial for our understanding of ScummVM because it allows us a more stable and sturdy starting point for an expansion of ScummVM. This means that less work and thought goes to waste by parsing out the GUI-related items within the rest of the code rather than straying ever further away from our conceptual architecture.

Expansion

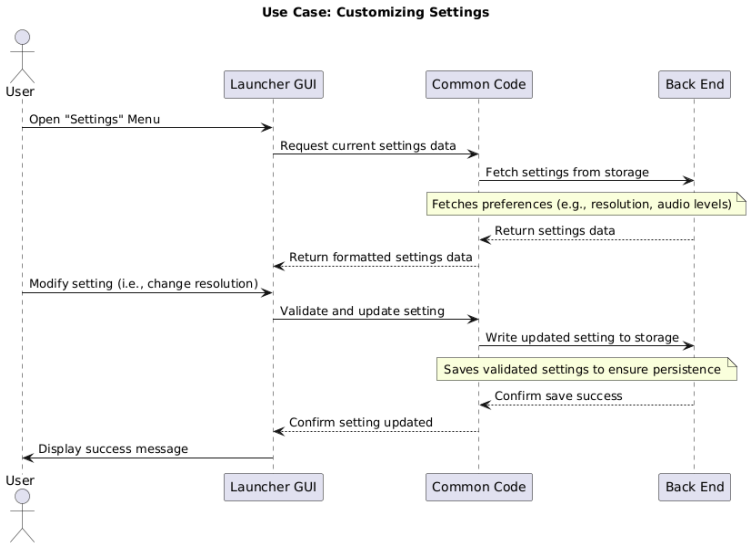
The biggest benefit of separating the GUI-related files, code and elements from the remainder of the ScummVM software is the opportunities that come about for expansion. Due to the disorganized nature of the code for ScummVM's GUI, it can be incredibly difficult to add new features or change pre-existing ones owing to the large number of dependencies that exist between the various subsystems, especially in relation to the GUI-related files. This issue is very handily solved by separating the GUI-related items from the larger body of code. By having the GUI-related items contained within the Launcher GUI subsystem, we can work to limit the number of dependencies overall which reduces the ripple effect of changes to the GUI, making it easier to change. Similarly, the Launcher GUI subsystem's dependencies on other large-scale subsystems could be minimized and more finely managed. Separating the code to create the Launcher GUI subsystem would ultimately

allow for more careful, fine management of the subsystem which would allow for the documentation to be built up more specifically around the GUI component. This will then further allow more developers greater insight into ScummVM's code and, thus, the ability to make more substantial additions to the, admittedly, lackluster GUI of ScummVM. Keeping in mind that the last time that the GUI received a major overhaul was 2009, for the launch of the 1.0 version of ScummVM, this update is sorely needed and the current GUI is certainly showing its age.[1]

So, by separating the GUI-related elements into their own Launcher GUI subsystem, we ultimately make the GUI easier to work with, allowing more and better contributions to that piece of ScummVM.

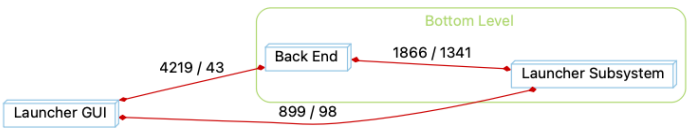
Required Modifications

The proposed enhancements largely serve to bring the concrete architecture of ScummVM closer in line with the conceptual architecture we proposed in our original report. To this end, a majority of required modifications relate to reorganizing and rewriting files and processes to better streamline data flow between different layers. At the high level, the concrete architecture of ScummVM would stay the same, with the main subsystems still being the Bottom Level, the Common Code, the Engines, and the Launcher GUI. Changes across the application would place an emphasis on the division of responsibility between different subsystems, especially between the Bottom Level and the Launcher GUI.



Example Sequence Diagram: Customizing settings within enhanced architecture

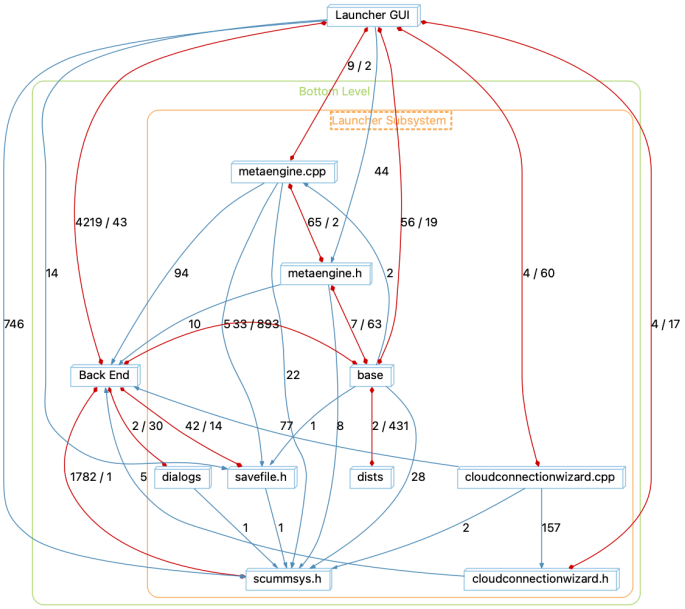
Bottom Level



The main change concerning the bottom level is that some of its functionality will be moved out to the launcher GUI. The main dependencies of the Launcher GUI on the Bottom Level currently include accessing game saves, and communication with the central ScummVM application loop. The Launcher GUI also has many dependencies on the Bottom Level which relate to simple calculations and data structures. This includes storing elements in arrays and drawing rectangles on the viewport. Depending on the degree they are used throughout the rest of the application, some of this functionality could be reassigned to be the responsibility of the

[1] https://docs.scummvm.org/en/v2.8.0/help/release.html#rc1-grog-xd-2009-08-31

Launcher GUI. As the Launcher GUI is the way the user saves and loads their game state, our proposed enhancement would also require reorganizing the way the application accesses user data. Rather than incorporating save game functionality into the back end, with the Launcher GUI having the capacity to access saves via the Bottom Level, saves could be accessed via the Launcher Subsystem, with other components accessing saves by making requests to the Launcher Subsystem, and the Launcher Subsystem placing requests to the Back End. This would place an emphasis on the interactions between the Launcher Subsystem and the Launcher GUI, and also emphasize the role of the Back End in communications with the OS (Operating System).



A majority of the necessary changes to the Bottom Level are specifically to the Back End subcomponent. The Launcher Subsystem has a clearly defined role as the base of the entire application, and as such there are no elements within it with a dedicated relationship to the Launcher GUI. In fact, most dependencies from the Launcher Subsystem to the Launcher GUI are due to the interaction of user actions and back end system calls (e.g. the cloud connection wizard). The only way to minimize these dependencies would be to remove the division between the Launcher GUI and the rest of the Launcher Subsystem, which would create more divergences from our proposed conceptual architecture for ScummVM.

Common Code

The main dependencies the Launcher GUI currently has on the Common Code relate primarily to graphics. Other functionalities which involve a significant amount of dependencies from the Launcher GUI to the Common Code relate to keymapping/input from the keyboard, audio processing, and accessibility options like text-to-speech functionality. The Launcher GUI and in-game overlay GUI could potentially benefit from dedicated versions of these resources, with the Common Code retaining ownership of versions specifically designed for non-GUI related functions. Although the Engine layer and the Launcher GUI both rely on many different elements of the Common Code, there are multiple cases where only one of them actually has dependencies on those elements.

Engines

The Launcher GUI and related elements of the ScummVM application operate largely independently of the Engines layer. On the other hand, there are many dependencies from the Engines layer to the Launcher GUI, most of which are related to the in-game overlay GUI which is available to users running a game using ScummVM. As a component of the ScummVM application, the in-game overlay is poorly defined in the current concrete architecture, and the main modifications required with regards to the Engines layer are to clearly organize and package elements related to the in-game overlay into distinct components.

The main components of the Launcher GUI which interact with the Engines that would not be changed in the proposed enhancement all relate to debugging. Both top-level files in the Engines layer and specific engines have many dependencies on debugging-related files part of the Launcher GUI subsystem, but as debugging does not constitute a core part of the main use cases of ScummVM, our proposed architectural changes would leave these files untouched.

Potential Drawbacks

There are several drawbacks of changing the architecture of any system. To begin, it's expensive, the required effort will be a lot to change how systems and subsystems depend on each other. There's also a risk of introducing or reintroducing errors on previously working functions. Finally there is a potential to complicate current and future functions.

Cost

The required refactoring of many existing systems in order to meet the new architecture requirements will cost developer resources in order to meet the changes. This means going into each dependency individually and confirming whether changing the architecture will affect these systems and how they operate. When something needs to be changed, it requires both changes in the code and the documentation. This will require thorough investigation for developers to spend development resources in making these changes and revisiting documentation to make sure it stays up to date with the new implementations. While making these changes there is an additional cost that new features have to be extra cautious about relying on any old implementations. This is because they might be changing and therefore development in other areas have to come to a halt while these changes are being made or might require extra resources now or later to be implemented correctly.

Risk

There are many risks when making structural changes of an existing system. Changing how systems are connected gives room for new errors to be created and for old errors to resurface. Maintainability is negatively affected while changes are being made, potentially introducing inaccuracies into testing due to the structural changes being in a state of limbo while they are in the process of being implemented. This also includes bug fixing since errors might arise from changes or the incompleteness of these changes. ScummVM supports a vast array of classic game engines, some of which rely on legacy interfaces. The decoupling of the GUI subsystem may inadvertently break compatibility with certain engines or require significant resources to maintain support. Changes to the pipeline of information through new layers of abstraction that are created from separating these subsystems also has the risk of worse performance, since new information may have to travel through more layers, possibly causing slower responses.

Complications

Existing subsystems are tied together through dependencies in ways that are integral to each other and work as they are. Changing this by separating them and potentially adding more steps in processes is inherently more complex as it increases abstraction. While this increased complexity may be easier to understand due to compartmentalization, it may also cause the inverse effect on new developers as how these systems are connected possibly becomes more convoluted.

SAAM Architectural Analysis

Identifying the Major Stakeholders

The enhancement being proposed for ScummVM's architecture has several key stakeholders with different priorities:

- ScummVM developers operate and extend ScummVM, with modularity and ease of extension in the future.
- Game players use the games, and will want to have a hassle-free, accessible gaming experience with no lag and minimal platform incompatibility.
- Engine developers add new game engines to ScummVM, and will need scalability as well as cross-platform functionality.

Identifying the NFRs

ScummVM developers prioritize Reusability and Performance. The system should be modular and adaptable, allowing components to be reused in different contexts without significant refactoring. The code should remain flexible for future changes, ensuring ease of extension without causing unnecessary overhead. Performance is crucial as any increase in complexity or inefficient code could negatively affect the overall speed and responsiveness of the system.

For game players, key NFRs (Non-Functional Requirements) are Performance and Portability. For gamers, the experience needs to be smooth, with smooth gameplay; thus, Performance is crucial to avoid any lag, delays, or frustrating transitions. This is where portability ensures that ScummVM runs on an extensive range of platforms from high-end devices to older or less powerful systems with wide accessibility and consistent quality of experience.

Engine developers focus on Scalability and Portability. As new game engines integrate into ScummVM, the architecture must support easy extension without major rewrites being necessary. Scalability allows for future engines to still be added in a timely manner without placing undue limitations. Portability is an equally important factor, wherein the system must function across diverse hardware and operating systems, including a wide range of devices and environments.

Method A

Method A proposes a single system that can adapt modes, from the launcher to/from the in-game overlay, based on context. This approach integrates both into one system, enabling it to simplify the architecture and minimize code duplication. The result of this is the fact that the coupling between the launcher and the in-game overlay may lead to a decrease in modularity due to the tight linkage of the two components. This may reduce the possibility of reusing parts of the code in other contexts or systems.

In terms of performance, mode switching may introduce delays at transitions and may affect user experience. The system would need to be managed for performance overheads, particularly on low-end devices.

Offsetting these two possible disadvantages, Method A may foster a better cross-platform feel because it uses one system rather than dealing with the added complexity of ensuring both pieces play nice on multiple platforms.

Impact on NFR Requirements:

For ScummVM Coders, Reusability may face potential issues because of the tight coupling of the launcher and in-game GUIs. The complexity of the system might impede code reuse. Regarding Performance, mode switching could lead to delays and would need careful management of resources to minimize the impact.

Game Players may experience hiccups during transitions that may be exacerbated on underpowered hardware, affecting Performance. However, Portability can be enhanced by lightening the overhead of multi-platform consistency.

Engine Developers may find Scalability an issue as the system's interconnected GUIs may impede future engine integrations. However, Portability benefits from reduced code duplication, easing the burden of cross-platform support.

Method B

Method B proposes that the system for accommodating the launcher and in-game overlay should continue to be separated into multiple sub-components within a larger GUI system. This would mean the launcher and in-game overlay remain separate, maintaining modularity. However, since some code and functionality would be shared between the launcher and in-game overlay, it would require that another sub-component be added to the GUI system to allow code reuse and reduce unnecessary duplication. This emphasizes modularity and reduces the amount of work that would need to be done to convert the existing system, but sacrifices some of the elegance that would come from the centralized architecture proposed in Method A.

Importantly, the current architecture for ScummVM already separates the launcher and in-game overlay GUI. Although we aim to reduce this split, keeping them separated reduces the amount of work that would need to be done to convert the existing system to the proposed system, reducing the time spent and minimizing unforeseen problems.

However, Method B would not simplify the architecture as much as Method A would. Because the sub-components would still be kept separate, some of the main advantages of combining them into one component are lost. However, since there is still reusable code within the component, it would still reduce the number of external dependencies.

Impact on NFR Requirements:

Method B impacts stakeholders differently than Method A, particularly with regard to ScummVM developers, who would likely have a much easier time implementing Method B than Method A. This is due to similarities between Method B and the current implementation, thanks to pre-existing modules. This means they would need to do less work, potentially shortening the project time and providing the finished project sooner. However, since this method might lead to more dependencies than Method A, it could also make the codebase less modifiable in the future, which could negatively impact the modifiability and modularity for ScummVM as a whole.

It is unlikely that the players would notice much of a difference in performance when compared to the current implementation of ScummVM’s launcher and in-game overlay because this method would not add any new functionality, instead focusing on the architecture and layout of the project.

Engine Developers would likely have the same problems as in Method A, since all the same issues with engine integration would affect them in the same way. However, since there is less code reuse in this method than in Method A, this could cause even more confusion and lessen the benefits explained earlier.

Determination 

Although both methods have their benefits, we think that Method B is superior thanks to its simpler implementation and shorter estimated project time. Because it requires less changes to the code, it minimizes the potential drawbacks explained earlier such as restructuring and changing dependencies. However, it still allows for increased code reuse and makes for a more intuitive architecture. Overall, Method B is a less risky change that still accomplishes our goals while requiring fewer code changes.

External Interfaces

The following list includes all items that were previously listed as external interfaces for the ScummVM software:

- Operating System
- Launcher GUI & Menu Dialog
- Game Engine GUI(s)
- External File Systems (e.g. Google Drive, OneDrive)
    - In order to load/save game data
- LAN / other networks for file sharing

Operating System

The redistribution of ScummVM to create a component more intently focused on the GUI of the software would have little effect on its relationship with the OS as a whole. Some portions of the new Launcher GUI component we are proposing would be coming from the Bottom Level layer, which we understand to be the component that has the most connection to the Operating System of the device on which ScummVM operates in a given instance. Additionally, much of the graphical components of ScummVM do rely on the Bottom Level layer for direct support with the OS. Thus, the Launcher GUI component will rely fairly heavily on the interface between the larger ScummVM system and the OS.

Launcher GUI & Menu Dialog

Clearly, we anticipate the largest change in the relationship between ScummVM and the Launcher GUI. Specifically, when discussing separating the Launcher GUI elements (which were previously strewn across the architecture's various components) into their own dedicated component. Clearly, the Launcher GUI's and the Menu Dialog's relationship with the entire system will change as it is pulled from wherever it may have been found before and placed into a previously non-existent component intended explicitly for files of their nature. The relationships between all of the architecture's components will change as they must now incorporate the loss of both significant portions of their code base and the simultaneous addition of an entirely new component composed of the previously lost code. Contrarily, the new Launcher GUI component will have entirely new and unique relationships to define between each of the previously existing components. The vast majority of this impact on the interrelationships of ScummVM has been explored throughout this report.

In terms of impact on use cases, the addition of a dedicated Launcher GUI component will have a similarly striking change to the overall aesthetic and functionality of the launcher GUI. As explored in the section on the benefits of this change, the separation of the Launcher GUI elements from the larger ScummVM system will allow further development of features, documentation and overall provide a stronger architecture to build from further. Thus, the aesthetic of the GUI can receive a significant overhaul, it can receive further feature updates to make its integration with other external interfaces easier and more accessible and developers can further focus on NFRs of ScummVM such as the portability of the system without compromising. This will, clearly, have a pretty significant impact on nearly all use cases given that nearly every user that utilizes ScummVM has to, at some point, access the software's GUI.

Game Engine GUIs

We expect that the decoupling of the Launcher GUI from the larger ScummVM software into its own component will have the second greatest impact on the Game Engine GUIs. Notably, both of these pieces of ScummVM concern the GUI of the software. However, beyond that, this will also provide an opportunity to better separate the Launcher GUI from the Game Engine GUIs. This being an issue that seems to have impacted much of ScummVM's GUI potential given the clear reliance of the software on the GMM (Global Main Menu). With the clearer separation of the Launcher GUI from the rest of ScummVM's systems, this similarly opens the door to greater clarity on the Game Engine GUIs. This will, similarly, allow for further development of the Game Engine GUIs to have a unique and more independent existence from the Launcher GUI component. If this were to come to pass, this would clearly have a great effect on the use cases of most users of ScummVM, given the strong reliance of users on the GUI of a software.

External File Systems

We anticipate that our proposed change to the ScummVM software architecture will have very little effect on the functionality of this component. As we are simply removing the GUI-related items from our other identified components and transferring them to their own dedicated component, we anticipate no unique complications arising in relation to external file systems. The most significant change that we would expect for this functional piece of ScummVM is an updated GUI.

LAN

As our proposed change has little effect on the functionality of much of the ScummVM software, the change would, similarly, have little effect on the relationship between ScummVM and LANs. Similar to the previous section on external file systems, the most significant change to ScummVM's relationship with LANs would be the opportunity to update the GUI that relates to its LAN file sharing feature. Overall, this external interface will be left largely untouched by this change, which simplifies the complexity of such a large change to the architecture of the software.

Conclusions

This report discussed the possible architectural modifications to the ScummVM GUI system, which are focused on enhancing modularity, scalability, and maintainability by decoupling the Launcher GUI from the greater software structure. Two methods were considered: Method A, which suggests merging the launcher and in-game overlays into one system, and Method B, which favors developing a separate, modular Launcher GUI component. Both methods offer distinct advantages, but the analysis reveals important insights into their implications for the system.

Method A reduces some of the overhead by including the combination of launcher and in-game overlays. It will, however, be more costly to implement and more risky due to the complexity of their integration. The potential for instability or inefficiency presents a significant risk, and the lack of clear separation between the launcher and in-game functions might hinder the scalability and flexibility of the solution in the long run.

In contrast, Method B is more balanced. It creates a separate GUI component for the Launcher, providing a moderate increase in code reuse and modularity at the expense of stability. A clear separation between the launcher and the rest of the system reduces risk, makes development easier, and makes the software adaptable for improvements in the future.

Method B's modularity allows the system to be scalable in the long run without losing its maintainability that is usually necessary for further development.

Both methods were assessed against the NFRs of ScummVM, which include performance, portability, and reusability. Method B is closer to these priorities because it introduces fewer structural changes and preserves the core strengths of the software, such as cross-platform compatibility and ease of use. While Method A could offer some efficiency benefits, the risks involved make it less suitable for achieving the desired improvements without compromising stability.

Regarding the external interfaces, such as file systems, operating systems, and LAN features, the changes are not expected to cause major disruptions. The associated GUIs may be adjusted, but the basic functionality will remain the same, with possible minor enhancements in terms of usability and user experience.

Ultimately, Method B emerges as the most feasible solution. This approach best balances the architecture's advance while minimizing risks, thus allowing ScummVM to continue its evolution in such a way that it increases modularity, maintainability, and cross-platform functionality. This will lay the best foundation for any future development while retaining the stability and accessibility that the users and developers of ScummVM rely on.

Lessons Learned

In evaluating the different considerations relevant to enhancing ScummVM as an application, we learned more about the factors which affect the software development process. By exploring the ways our proposed enhancement would affect the software at every level, it was easier to see the dependencies and interactions which most greatly impact the software. Even though our proposed enhancement was focused on only the Launcher GUI, we uncovered many interdependencies which branched across the different high level subsystems present in the architecture.

Similarly, by outlining our plans and suggestions for such a structural change to ScummVM, we have further realized the complexity and difficulty of even a small structural change. Throughout this report, we have examined many of the different impacts that the addition of a single new component to ScummVM would inspire in every other high-level subsystem that we had previously identified. This has cemented in our minds the importance of beginning a software development project with a clear and concise plan of action because it was significantly easier to dream up a conceptual architecture than it has been to discuss the full ramifications of adding a single new component to the concrete architecture.

Through the process of exploring different methods of implementing our proposed enhancements, we were able to highlight related non-technical factors, and specifically how they would impact its development timeline and exact implementation. We were forced to consider the needs and constraints coming from different stakeholders, including the software development team for ScummVM, developers for a ScummVM engine, and the average user, all of which interacted differently with the core changes related to the enhancement.

In addition, we have also begun to realize the possibilities of different identified architectures of a software architecture like ScummVM. Of course, we have seen many differing architectures throughout this course during our feedback and such, but this has been a slightly different perspective on that experience. Having our completed concrete architecture and being asked to come up with something to add or change about it to improve the software as a whole has opened our eyes to the possibilities of different layouts of architecture. To think that ScummVM could have already been built in such a way, or all of the many features that other groups on this project must have dreamed up. Not only that, but the possibilities of what could be accomplished with such a small (albeit, complex) shift in

the architecture of the software are endless. We've found all of this rather inspiring for our future endeavours in computing.

Overall, proposing an enhancement to a specific software system necessitated a deeper understanding of both the software itself (ScummVM) and the software development process in general. The most important lessons learned from this assignment was how important it is to consider any changes to a piece of software holistically, with reference to both individual components of the software and the functioning of the software as a whole.

Naming Conventions / Glossary

GMM: Global Main Menu. This refers to the menu a user can call while playing a video game using the ScummVM software. Its primary functionality is to save the game and load the game using said saves as well as to return to launcher/quit.[2]

LAN: Local Area Network. A local area network is a network that connects devices throughout a localized area.

NFR: Non-Functional Requirement. A non-functional requirement is, exactly as the name states, a requirement of the software that is not strictly necessary. Examples include portability, modifiability or performance.

OS: Operating System. An operating system is a piece of software which runs on a given machine which allows processes to display output, access files, and interface with other hardware resources in an organised fashion.

SAAM: Software Architecture Analysis Method. As the name suggests, this is a method that is used in the analysis of the architecture of various softwares. It is particularly interested in NFRs.

UI: User Interface. This is the part of the system that the user can see and interact with. Can be a Graphical User Interface (GUI) for visual navigation, or a Command Line Interface (CLI) for navigation using text commands.

References

ScummVM Development Team. "Release Notes." *ScummVM Documentation*, 2023, docs.scummvm.org/en/v2.8.0/help/release.html#rc1-grog-xd-2009-08-31. Accessed 1 December, 2024.

ScummVM Development Team. "Saving and loading a game." *ScummVM Documentation*, 2023, https://docs.scummvm.org/en/v2.8.0/use_scummvm/save_load_games.html. Accessed 1 December, 2024

[2] https://docs.scummvm.org/en/v2.8.0/use_scummvm/save_load_games.html