CISC 322/326 - Software Architecture
Assignment #3: Report

Architectural Enhancement for ScummVM

TC++
Evan Cook - 21egc10@queensu.ca
Hugh Tuckwell - 21halt@queensu.ca
Will MacInnis - william.macinnis@queensu.ca
Liv Stewart - olivia.stewart@queensu.ca
Jay Wu - 21jjw12@queensu.ca
Nate Rinsma - 21nr6@queensu.ca

Table Of Contents

1. Abstract

In deliverable three, we will be proposing an enhancement to the ScummVM software and how it is incorporated within the architecture. Specifically, this report proposes the integration of an AI-powered hint system named Scummy into the ScummVM architecture. Scummy would enhance the user experience by providing seamless, in-game assistance tailored to players' specific needs. Users will be able to quickly access hints with a single click, instead of having to open an external browser. This enhancement keeps the user engaged and provides the resources for a smoother gameplay experience. Scummys goal is to help players when they come across challenges within the game by providing context-specific giants generated through AI models like ChatGPT and Llama. For example, when a player is stuck, they would click on the Scummy icon and Scummy would generate a hint based on their current in-game scenario.

Implementing Scummy creates the need to adjust the architecture from the last report, by adding a Scummy Module component, dependencies on already existing systems and updates to the Launcher to enable AI interaction.

Additionally, the non-functional requirements of portability, security, accuracy, performance and maintainability are required to provide a strong, dependable and easy-to-use implementation. This additional feature will benefit the users by improving their overall gaming experience as well as developers by making sure the feature complies with the open-source guidelines and standards of ScummVM.

We will discuss potential risks of implementing this feature which include privacy concerns due to ChatGPT requiring an active internet connection for functionality and possible exposure of user data. The Llama implementation, while offline, may impose a high CPU load and face challenges with maintainability due to potential updates. This is why ChatGPT was ultimately deemed the better implementation due to its portability, performance, ease of access, and accuracy.

To ensure reliability, both unit and integration testing will be done. Unit tests will verify core functions like screenshot capturing and API response handling, while integration tests will replicate complete user flows to confirm smooth cross-platform execution. Various testing scenarios include handling errors and macOS screen recording permissions.

We also will talk about the key stakeholders, including users and developers, and will focus on usability, accuracy, ease of installation, and maintainability, ensuring that Scummy delivers a valuable and robust experience for the entire ScummVM community.

2. Introduction

ScummVM is a well-known open-source software that lets people play vintage point-and-click adventure games on all devices. The main objective is to provide players with a seamless experience across many platforms and operating systems, while still delivering an accurate representation of the vintage games. One of the challenges users face is getting stuck on difficult or complex problems as well as needing clarification on obscure gameplay mechanics.

Players would usually turn to external walkthroughs and sources for aid, which ultimately disrupts their gaming experience. To address this issue this report delves into an innovative enhancement within ScummVM.

In this third report, we will be introducing a new feature to ScummVM. This feature involves the use of AI to create a hint system for players called Scummy. This AI will use either ChatGPT or Llama. Players will be able to interact with Scummy by providing screenshots to be given hints as to what task to perform next. This feature's goal is for convenience when players are stuck and require help without needing to rummage through online sources for walkthroughs.

With the addition of this new feature, ScummVM's architecture will be changed accordingly. A new AI component will be added in the form of a new component called the Scummy Module. The Scummy Module is a dedicated component for Scummy's functionality, as integrating it into existing components like the Common Component would bloat them unnecessarily and create unintuitive dependencies. This separation ensures clarity and maintains the modularity of the system. With the addition of this component, there will be new dependencies introduced. The Scummy component will depend on the Common component for utility functions, the OSystem API component for handling OS-specific tasks like taking screenshots and sending API requests, and the Graphics component for rendering the Scummy icon. Likewise, the Launcher and OSystemAPI components will depend on the Scummy Module to manage the user input and test functionality during setup. These dependencies ensure seamless integration and functionality of Scummy within the system.

Now that the AI component is added to the overall architecture, it's important to visualize and describe how it would be used with the other components. We have provided a use case diagram illustrating a scenario where the user asks Scummy for an in-game hint. This process sequence involves clicking the Scummy icon mid-game to request a hint, to display a popup. After the user clicks "I need help," a screenshot is sent to the AI (ChatGPT or Llama), the response is displayed on-screen, and the popup is closed with a "Thanks Scummy" click, returning to the original icon. There is also a second sequence diagram illustrating a scenario where a user enables Scummy on MacOS. This use case outlines the required elevated permissions needed on systems like the macOS. It further shows how the user activates Scummy through the Launcher, grants screen and audio recording permissions in System Settings, and confirms functionality with a test request, after which the Launcher saves the enabled setting.

Furthermore, this AI feature needs to have many important non-functional requirements. These include portability, security, accuracy, performance, and maintainability. Portability is important as Scummy is expected to work seamlessly across all platforms where ScummVM is supported. Since ScummVM is known for its high portability, it must maintain this standard for Scummy as well. Security is essential to protect user data and ensure safe interactions with external AI models and services. Accuracy within Scummy ensures that users are receiving helpful, context-specific hints that are relevant to what they are currently doing in the game. If Scummy cannot produce accurate and correct responses, then users will turn to external sources,

which defeats the whole point of Scummy. Performance is needed to provide users with fast response times requesting hints. Having delays in generating hints would interrupt their gaming experience, and yet again, would make the Scummy Module useless. The Scummy Module must be maintainable so that it can be easily updated, debugged, and improved throughout the years.

When adding this feature, it's important to address who the stakeholders are. The two main stakeholders for the change are users/players and developers. Of course, since the players are the ones who are going to be interacting with Scummy during gameplay, it's important that the feature meets their demands and doesn't negatively affect their experience. The developers are also stakeholders as the product's reputation is on the line, so they want it to be well-made.

We considered two implementations for Scummy, one method using OpenAI's ChatGPT and the other using Meta's Llama as the AI model. The ChatGPT approach requires users to set up an OpenAI developer account and input their API key, allowing ScummVM to leverage ChatGPT for hints without burdening the development team with costs. Meanwhile, the Llama approach involves users downloading and running a local instance of Llama, adding complexity by requiring independent setup and verification, creating a higher barrier to entry. With these different implementations, we compare the important NFRs (i.e. portability, security, accuracy, performance, and maintainability) to help determine the pros and cons of each method.

That being said, the implementation of Scummy is not without potential risks. Comparing ChatGPT to Llama, we considered how each method impacts ScummVM. We address risks like the implementation of Scummy with ChatGPT which requires an active internet connection and poses privacy risks, as screenshots sent for analysis could expose sensitive information. On the other hand, using Llama avoids this issue by running locally, however, it demands significant CPU resources, potentially limiting functionality on lower-end systems. ChatGPT offers easier maintainability with simple API calls but risks downtime or server overload, causing reliability issues. Conversely, Llama updates could introduce instability, adding another layer of unpredictability to its use.

After evaluating both implementations, We believe that ChatGPT emerges as the better choice for Scummy due to its portability, aligning with ScummVM's functionality. It also offers better performance, ease of use, and maintainability compared to Llama, which requires significant training data to achieve accuracy. While Llama avoids privacy concerns by running locally, its initial inaccuracy and high CPU demands make it less viable. ChatGPT's superior hint accuracy and accessibility make it the clear choice.

To determine how well the new AI feature will be, we suggest unit testing and integrating testing strategies. The unit tests will validate individual components, such as screenshot capturing, prompt formatting, and API response handling, using mock APIs to simulate various conditions. Meanwhile, integration tests will evaluate complete user flows, including macOS-specific scenarios like screen recording permissions, ensuring seamless functionality and user experience. These methods of testing will ensure the robustness, maintainability, and usability of Scummy.

3. Proposed Enhancement

Our proposed enhancement for ScummVM is an in-game hint AI-powered hint system, called Scummy. How this would work is when the user is stuck on a game, they can click on the Scummy icon in the corner of their game, and it will generate a hint for them based on where they are in the game. An AI model will generate the hint, either ChatGPT or Llama depending on the implementation, by sending a screenshot along with a prompt to generate a hint for what to do based on the screenshot. The generated response will then be displayed to the user. Our goal with this enhancement is to provide a better user experience, where instead of users having to leave ScummVM to search through a wiki to figure out what to do next, they can get an answer quickly and easily all within ScummVM.

3.1 Architectural Changes

In making this enhancement, we will be required to update our conceptual architecture so it properly reflects our new feature. For reference, the current conceptual architecture is displayed below.

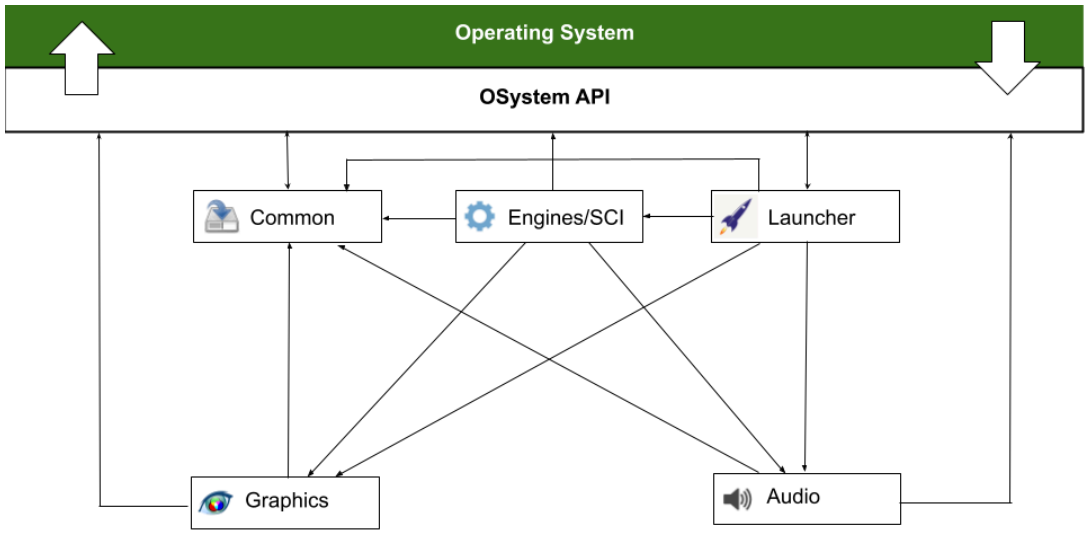

Figure 1: Current ScummVM Conceptual Architecture

For our new feature, we proposed a new component to handle all of the functionality related to Scummy, called the Scummy Module. We came to this decision because it doesn't make sense for any of the other components to contain the functionality of Scummy. The component that could be considered to contain Scummy could be the common component, however, this is more reserved for utility functions for other components and Scummy is distinct enough that we believe it should have its own component, and placing it in common would lead to common being more bloated than it needs to be, as well as introduce unintuitive dependencies.

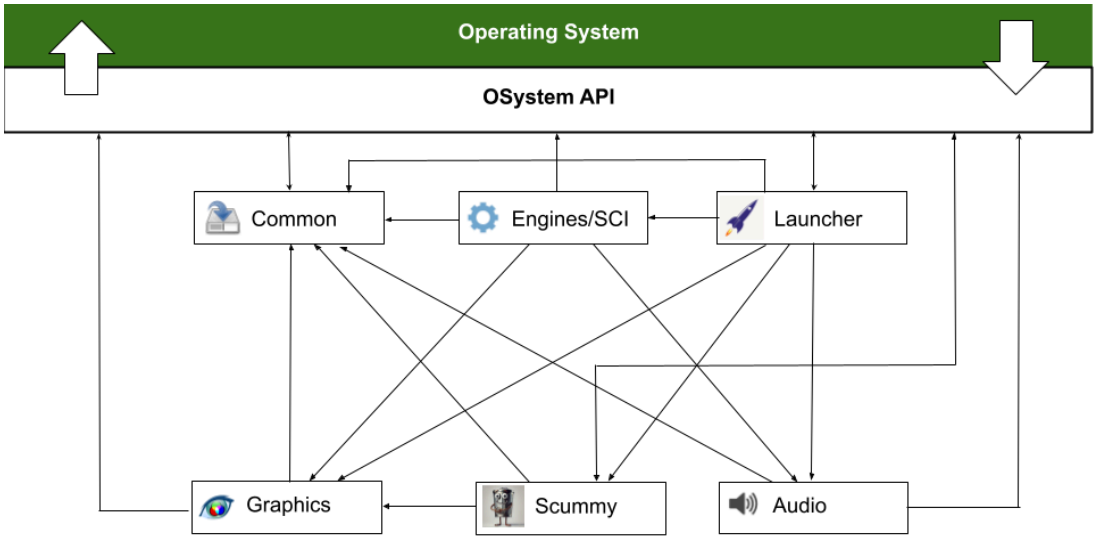

Figure 2: New ScummVM Conceptual Architecture

As the Scummy Module is a new component, it introduces new dependencies to the conceptual architecture. The Scummy Module depends on the common component, OSystem API, and graphics. It depends on the common component for the same utility functions that all the other components depend on. It depends on the OSystem API because it needs to request a screenshot to be taken of the game, which is operating system specific and thus must be handled by the OSystem API. The Scummy Module also depends on the OSystem API because it needs to be able to send an API request to the AI model, which is an operating system-specific task. The OSystem API must also be updated to include these functionalities. Lastly, it depends on the graphics component because the Scummy Module needs to be able to render the icon for Scummy so the user can interact with it, so this will be handled by the graphics component. The Scummy Module is utilized by both the Launcher and the OSystem API. The OSystem API relies on the Scummy Module to handle user input, calling its functions to enable seamless interaction with Scummy. The launcher component depends on it because when enabling the use of Scummy in the launcher settings, the launcher sends a test request to the Scummy Module to ensure it has been properly enabled.

3.2 Use Case Diagrams

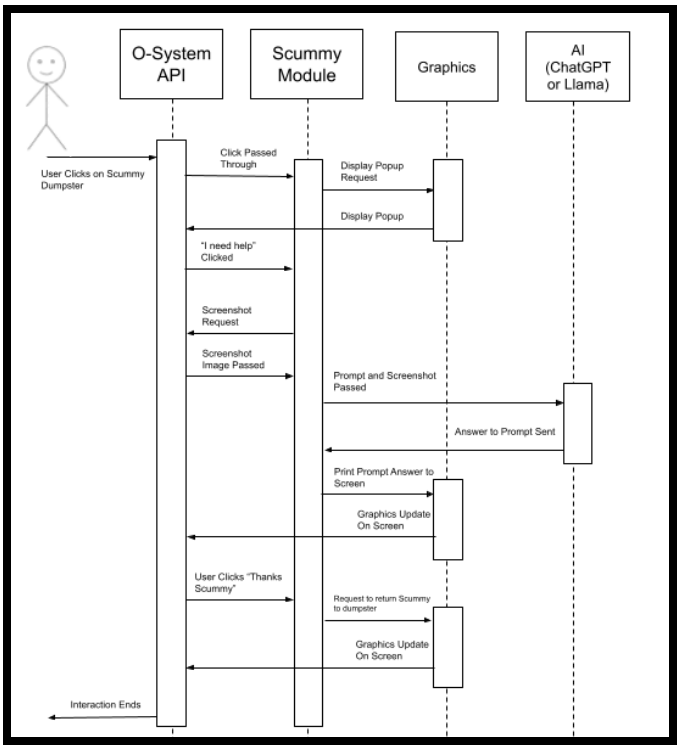

Figure 1: Use case 1, A user asks Scummy for a hint in-game.

This use case demonstrates a user asking for a hint mid-game. The user clicks the dumpster icon on the screen, the click request is passed by the O-System API to the Scummy Module. A request to Graphics is then made to display a popup. Once the “I need help” button is clicked a screenshot request is sent to the AI endpoint, either ChatGPT or Llama. The answer to the prompt is sent back to the Scummy Module, then the prompt answer is printed on the screen using the Graphics Module. The user then clicks “Thanks Scummy” to hide the popup and return Scummy to the dumpster icon.

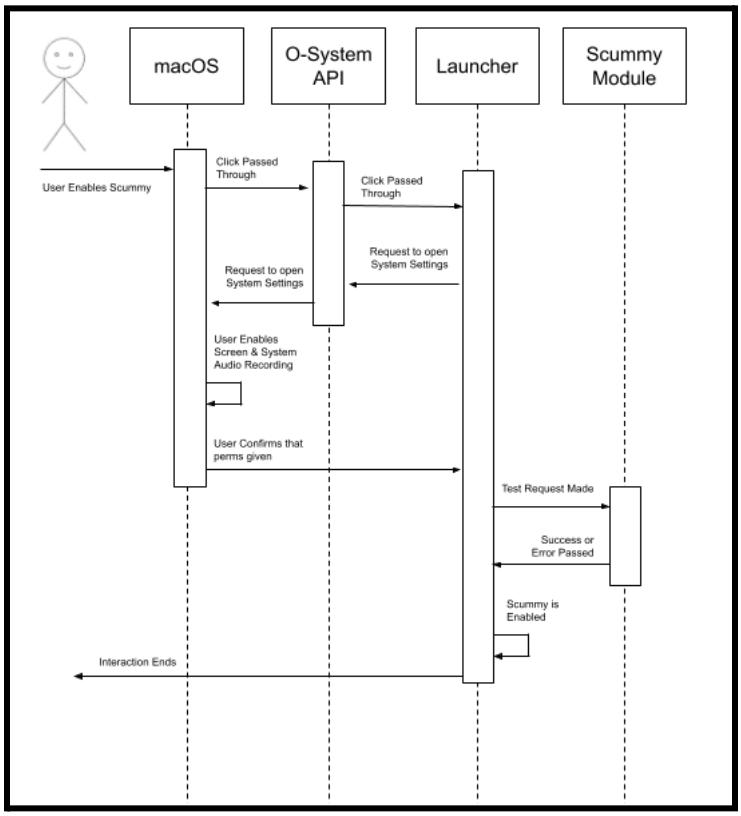

Figure 2: Use case 2, A user enables Scummy on macOS.

This use case demonstrates a user enabling Scummy on macOS. macOS has higher permission requirements for applications than Windows or Linux making the process more complicated, which is why this use case is important. Scummy is enabled through the Launcher using a button, a request is then sent to macOS to open System Settings. The user enables ScumVM to be able to record the screen and system audio. The user then confirms in the ScumVM launcher that the permissions have been given, and a test request to the Scummy Module is then made to see if the module is working correctly. Once a successful request is sent back by Scummy the launcher changes the game settings to remember that Scummy is enabled.

3.3 Non-Functional Requirements

The non-functional requirements we believe to be the most important for this enhancement are portability, security, accuracy, performance, and maintainability. Portability is an important non-functional requirement for this feature, as ScummVM is a highly portable software and this feature must be also highly portable so it can be used anywhere ScummVM is used. Security is important because we want to ensure that the software is secure, as well as to protect our users' data. There isn’t a ScummVM-specific reason for security, just that security is

a generally important thing to have in almost all software. Accuracy is important for our feature because we want to ensure that the users are provided with the most accurate advice possible by minimizing incorrect responses. Performance is important because we want to ensure that Scummy has a fast response time so the users aren't waiting too long for a response. Finally, maintainability is important as this is open-source software, so we want to ensure that the code can be easily maintained and easy to understand so new developers can read it and understand what it does and how it can be changed, and so current developers can easily make changes to it when necessary that doesn't lead to unexpected bugs.

3.4 Stakeholders

There are two main stakeholders involved with the new feature, the users and the developers. The users are stakeholders as they are the ones who will be using the new feature, and as such they want the feature to be well executed so it works well when they use it. They are mainly focused on whether the new feature works as intended and how useful the hits are in-game. Ease of use is also important to users, as if the addition is not easy to install users will not use the addition altogether. The developers of ScumVM and Scummy are also stakeholders as they are the ones required to develop the new feature, and as such they want to make sure their feature is well executed so their users are satisfied and to maintain the project's integrity by not releasing a bad product. Since ScumVM is an open-source project other developers may have to edit Scummy at any time. Scummy must be easy to edit and documentation must follow these requirements.

3.5 Implementations

The first implementation we considered was utilizing ChatGPT as our AI model to generate a response for the hints. How this would work is the user would set up Scummy by entering their ChatGPT API key into ScummVM, and allow ScummVM to take screenshots on their device. This way, the user can leverage their ChatGPT account to access the AI hint system, as opposed to the cost of operation being on the ScummVM development team. With this system a user would have to set up an OpenAI developer account themselves, and input their API key which decreases the ease of use significantly.

The other implementation we considered was utilizing Llama as our AI model. This would work similarly to the ChatGPT implementation, however instead of using the user's ChatGPT API key, the user would be required to download and run a local instance of Llama, which ScummVM would access when generating a hint. Like the ChatGPT implementation, this would also require the user to allow ScummVM to take screenshots on their device. This would mean that Llama would have to be downloaded separately by users, adding another barrier to entry to this implementation. Users would also have to verify that Llama is working independently of ScumVM, which would add a level of hoops a user has to jump through to get Scummy working.

3.6 Non-Functional Requirements for Implementations

Naturally, both implementations will have different advantages and disadvantages for each non-functional requirement, which we will go over now.

Portability: We talked about earlier how portability is highly important to the ScummVM project, as one of its key points is that it is compatible with a wide variety of systems. Due to this, portability will have a large influence on which implementation we go with. The ChatGPT implementation is highly portable, as it is compatible with any system with internet access. This is because ChatGPT is a remote model, so all that's needed is internet access to interact with ChatGPT. As for Llama, the user is required to download Llama, which is only available on Windows, MacOS, and Linux. Due to this restriction, the Llama implementation is highly unportable, restricting access to only three operating systems. Overall, the ChatGPT implementation is the clear winner in terms of portability.

Security: Security is obviously important for any sort of software, so as such we will discuss it here. For the ChatGPT implementation, the main security risk would be personal information getting sent to ChatGPT / OpenAI due to a potential bug where ScummVM takes a screenshot when it shouldn't, capturing personal information. This can be avoided with proper testing with the screenshot functionality. As for Llama, the main security risk would be the fact that the user is required to download additional software independent of ScummVM, that can't be verified by the ScummVM developers. Llama is open source, so it is likely highly secure, so the main security risk to the user would be from user error by accidentally downloading a virus during their attempt to download Llama. Overall, both implementations are quite secure, so this doesn't affect our decision too much.

Accuracy: What the user actually cares about at the end of the day is the hints that they receive from the AI model. ChatGPT is a more general AI tool that can access limited portions of the internet. This means that when a screenshot is fed to it, it can use the knowledge that it has from other gamers using the platform to answer the questions. The more ScummVM users use the platform the better ChatGPT will be at giving hints. On the other hand, Llama can be used offline but is less accurate for our use case than ChatGPT. This is because to get accurate hints from Llama we would have to train our own model to distribute to Scummy. Based on our current implementation of users installing Llama themselves, Llama would not have a lot of data to pull from. This would lead to inaccuracies compared to ChatGPT. If users want maximum accuracy they would need to go through the process of setting up a ChatGPT account.

Performance: Performance is also a non-functional requirement we care highly about, as we don't want to decrease the performance of ScummVM with the implementation of the new feature. For the ChatGPT implementation, the performance largely hinges on the user's internet speed, as well as the performance of ChatGPT. Since we can't control ChatGPT's performance, we will mostly consider the internet speed as the main performance factor for the ChatGPT implementation. For Llama, the AI model runs locally on the user's device, meaning the

performance hinges on the computing power of the user's device. Also, due to Llama using the user's CPU to generate hints, the performance of ScummVM could be hindered while Llama is generating hints. Due to the computing power likely being a bigger bottleneck for the average user compared to internet speed, and the fact that using ChatGPT doesn't take CPU usage from ScummVM like Llama does, the ChatGPT implementation will likely have better performance.
Maintainability: In regards to maintainability, our implementation relies on outside developers to maintain their codebases and documentation. Any changes made to Llama or ChatGPT's architecture will impact the usability of our addition. Developers working on our project will need to keep a close eye on Llama and ChatGPTs changelogs to make sure any changes do not impact our codebase. If our project is not maintained, it can cause internal processes to break, and the addition will become unusable. It is also possible that ChatGPT one day will remove the ability to query their API, which will also render our addition bricked. To make sure Scummy is maintained a team of developers must be devoted to keeping the project alive and thriving.

3.7 Potential Risks

One potential risk to implementing Scummy via ChatGPT is that there would need to be an active internet connection while ScummVM is running. The problem here lies in Scummy's ability to take screenshots of the user's desktop to be sent to ChatGPT for analysis. This would potentially allow a malicious individual to gain unauthorized access to information on a user's screen, compromising privacy. The implementation of Scummy by using Llama would not suffer this issue, as Llama is installed on the end user's machine and runs without internet. There's also a concern related to maintainability. With the ChatGPT implementation, there would simply be an API call to obtain the requested result; and an adjacent action for Llama. The problem though, is that there might well be an update to Llama somewhere down the line. Depending on what gets altered or removed with this hypothetical update, the answers obtained might be unstable or unpredictable. Llama itself would also be fairly CPU-heavy as a result of being installed directly on a user's system. Due to the large variety of systems that ScummVM is able to run on, there simply may not be enough processing power available to run the Llama implementation of Scummy efficiently, or at all. Reliability is also a potential problem, specifically regarding the ChatGPT implementation. For example, ChatGPT could be down or overwhelmed on a particular day. In that instance, a user could send a request through Scummy which would cause an infinite wait, as ChatGPT would never respond under those circumstances.

3.8 Better Implementation

After looking into how developing each implementation would work, as well as the impact the different implementations would have on the non-functional requirements, we believe ChatGPT would be the better implementation. The ChatGPT implementation is much more portable, which is basically a deciding factor given the existing functionality of ScummVM. Moreover, the ChatGPT implementation is likely more performant, and easier for the user to

access and maintain. The accuracy of the hints given is also a major deciding factor on which implementation we should go with. Due to the fact that Llama relies on data issued by the user to train itself, Llama may be unusable at first, needing a lot of training data before it becomes anywhere near accurate. Given this information, ChatGPT seems like the obvious choice for implementation.

3.9 Testing

To ensure the Scummy AI hint system functions smoothly and reliably, we propose implementing a testing strategy that utilizes unit tests and integration tests. Unit tests will focus on individual components, such as capturing screenshots, formatting prompts, and handling API responses, ensuring each function performs as expected in isolation. For example, the screenshot-capturing function will be tested to verify that it saves images in the correct format and location, and mock AI APIs will simulate responses to test the hint generation process under various conditions, such as valid hints, null responses, and errors. Likewise, Integration tests will evaluate how different components of ScummVM interact with the new AI feature. These tests will simulate complete user flows, such as clicking on Scummy, requesting a hint, and displaying the response, to ensure seamless functionality across modules. Specific tests will address critical scenarios, such as macOS permission handling for screen recording, ensuring users can enable Scummy without unnecessary hurdles. By combining unit and integration testing, we aim to validate the robustness, maintainability, and usability of the Scummy component, ensuring a smooth and reliable user experience.

4. Conclusion

Throughout this report our focus was on a new feature that could be added to ScummVM, being an AI hint assistant. This new feature would require the Scummy Module to be added to the overall architecture. This component would interact and depend on the Common, OSystem API, and Graphics components, while the Launcher and OSystem API components will depend on the Scummy Module. In response to this new module, we provided two sequence diagrams of how it would work, which are where the user asks Scummy for an in-game hint and a scenario where a user enables Scummy on MacOS. We then discussed the important non-functional requirements that are important to this feature, being portability, security, accuracy, performance, and maintainability. With the addition of this feature, we addressed that the two main stakeholders for this change would be the players and the developers. We discussed the variations for the implementation of the feature. Looking at both ChatGPT and Llama versions of the implementation, we addressed how each addresses the different advantages and disadvantages of the NFRs stated previously. The potential risks of both implementations were also addressed. With many factors combined, we concluded that ChatGPT would be the better implementation for the AI hint feature. Finally, we covered how the feature should be tested to ensure a seamless experience.

5. Lessons Learned

One of the key lessons that we learned while doing this report was understanding how to evaluate different implementation strategies by comparing their feasibility, impact on performance, and integration complexity. We also learned how to prioritize stakeholder needs, ensuring the feature enhanced user experience while maintaining ease of maintenance for developers. We learned that there are various ways to implement different approaches and choosing the best and optimal approach can be difficult especially when it is needed to follow many NFRs. We gained a deeper understanding of how new features interact with an established architecture, particularly in terms of managing dependencies with core components like OSystem API and Graphics. Overall, we gained a deeper understanding of how to design robust, user-centric features that align with the principles of open-source development.

6. Naming Conventions

AI: Artificial Intelligence
API: Application Programming Interface
CPU: Central Processing Unit
MacOS: Mac Operating System
NFR: Non-Functional Requirement

7. References

[1] *SCI. (n.d.). ScummVM :: Wiki*. https://wiki.scummvm.org/index.php/SCI
[2] *TCPlusPlus.org*
https://tcplusplus.org/documents/CISC%20322_326%20-%20Software%20Architecture%20Report%201.pdf
[3] *TCPlusPlus.org: https://tcplusplus.org/documents/A2%20Good%20Copy%20(1).pdf*
[4] "OpenAI Platform." *Platform.openai.com*,
platform.openai.com/docs/api-reference/authentication.
[5] "Documentation | Llama." *Llama.com*, 2022, www.llama.com/docs/get-started/. Accessed 29 Nov. 2024.