CISC 322/326 - Software Architecture
Assignment #1: Report

Conceptual Architecture Analysis of 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**
2. **Introduction and Overview**
3. **Derivation Process**
4. **Architecture**
    - 4.1. **Architecture Style**
    - 4.2. **System Overview**
    - 4.3. **Development History**
    - 4.4. **SCI**
        - 4.4.1. **Introduction**
        - 4.4.2. **Architecture and Functionality**
        - 4.4.3. **History and Evolution**
    - 4.5. **Control Flow**
    - 4.6. **Use Cases**
5. **Data Dictionary**
6. **Naming Conventions**
7. **Conclusions**
8. **Lessons Learned**
9. **References**

1. Abstract

This report provides a general overview of the software architecture of ScummVM which is a software project targeted at the emulation of classic graphical adventure games and role-playing games. What had started off with a codebase that had been crafted to support specifically the SCUMM games of LucasArts expanded to become a multi-engine platform able to run several thousand point-and-click games on quite a number of desktop and mobile operating systems.

ScummVM's architecture is designed with scalability and modularity in mind; such a layered structure means adding new game engines or function implementation doesn't cause too much disruption. While the entire system fits under a layered style architecture, its worth noting that at the base level, ScummVM is interpreter style based on the fact that it converts game-specific code into a format that is compatible with various operating systems.The major components comprising the system concern the following: Game Engines, Audio, Graphics, Input Handling, Management of the Game State, Launcher, and OSystem API. Each of these modules will play a critical role in ensuring cross-platform compatibility, smooth functionality, and user-friendliness.

Another critical addition will be the SCI engine: Sierra's Creative Interpreter, which will interact with these other components in managing the state of the games, input/output interfaces, and degree of audiovisual fidelity. In design, all classic older games will be preserved and adapted to modern systems without losing their original authenticity.

Thus, the report, in general, sums up that ScummVM is based on such a flexible architecture that allows and supports constant development and serves as an effective and strong basis for classic game emulation across different environments. This modular design allows for seamless updates as well as guarantees long-term relevance against the ever-changing landscape of gaming.

2. Introduction and Overview

Initially released on October 8, 2001, ScummVM, which stands for Script Creation Utility for Maniac Mansion Virtual Machine, was originally designed to run LucasArts adventure games on the SCUMM system. Over years of development and changes, ScummVM has become recognized as a set of game engine recreations that allows users to play certain classical graphical adventure and role-playing games from various non-SCUMM games.

It is analyzed that ScummVM follows a layered architectural style to effectively organize the functions required to emulate the games. Furthermore, this style allows ScummVM to scale with the addition of new game engines and features without affecting the capabilities of the system. While the overall system follows a layered style, the core of ScummVM follows an interpreter style that allows the system to run point-and-click games regardless of the platform it is running on. Together with the interpreter style, ScummVM's architecture includes modular components that interact with the game engine interpreter.

These components have been identified as 7 distinct modular components which are the engines (specifically SCI), audio, graphics, input handling, game state management, launcher, and OSystem API. To be more specific, the engine components like SCI are important for interpreting game data files and managing the audio and graphic components to create tangible visual and aural experiences. The audio component includes various drivers and encoders to ensure that games maintain their original sound quality by loading, mixing, and playing audio assets which are evoked by in-game events. The graphics component manages visual output by rendering both 2D and 3D game worlds along with the GUI to incorporate various scaling methods to adapt games for modern displays and various rendering models to emulate the appearance of the different systems. The input handling components register the inputs made by the user through mediums of their choosing. These inputs are then processed through the games' engines so that a wide range of systems and input devices are accommodated. The game state management component monitors the player's progress by tracking variable values and events. This component also features autosaving and manual saving which allows players to reload their game states through a Global Main Menu. The launcher component handles the various engines and offers a graphical interface for users to manage their games like loading, removing, launching, or even editing the games. These actions can be performed because the launcher communicates with the platform abstraction component to access these game files and load games with their corresponding engines into memory. Finally, the OSystem API component is the bridge connecting ScummVM with the operating system of different platforms. This platform abstraction component specifies the various features of games. It ensures compatibility across various platforms through interactions within the input handling, launcher, and auditory and graphical components to manage system resources.

This report also covers the major points in the history/development of ScummVM. This starts all the way from the beginning when Ludvig Strigeus wanted to create an engine for running Monkey Island 2 on Linux. The history then continues to announce how Vincent Hamm and many interested developers created and joined the team. Then the history mentions how the development team made significant rewrites for improved functionality and user experience. It also mentions the addition of the SCI engine. Finally, it concludes with the current state of ScummVM and how it is an open-source platform that supports a wide variety of games and systems.

Since various games require specific game engines to run, there are many engines that ScummVM can run. In this report, only one engine will be analyzed: the SCI engine. Similar to ScummVM, the SCI engine follows an interpreter architectural style with a PMachine as its core component. This PMachine is essentially a virtual CPU that allocates heap memory, has its own set of registers and a stack, reads instructions from data files, and executes the game based on the previous resources. We will also be talking about the history and evolution of the SCI engine.

This means starting from SCI0 which supported 230x200 graphics with 16 colours. Then evolving into SCI1, SCI 32, SCI 2, and SCI 3, which had increasingly higher-resolution graphics and expanding resource types.

We also provide a dependency diagram of the ScummVM architecture. This visually displays how the 7 modular components interact with each other and the operating system of the machine that runs ScummVM. We then further explain how each component may interact with its fellow components.

Following shortly after, we conceptualized 2 use cases of ScummVM. The first use case shows a sequence diagram of how the system would process and execute playing a point-and-click game on ScummVM. The second use case is also a sequence diagram but shows how the system would process and execute launching a game on ScummVM. Whereas the dependency diagram simply showed which modules interacted with what, these sequence diagrams show a concurrency view of how the different modules interact with each other and when/the order in which they would interact

Finally, we conclude the report with a summary of our key findings of ScummVM and proposals for future directions. With our findings and proposals in place, we reflected on the lessons we learned while conducting our research on the conceptual architecture of ScummVM.

3. Derivation Process

When we assigned ScummVM as our software of interest for the project, the first thing we decided to do was for each of us to do our own independent research and document all of our findings in a shared brainstorming document. Doing this gave us all a solid understanding of the function of the software as well as ideas for potential components and architectural styles. After conducting this independent research, we had a meeting to discuss our findings with each other to further strengthen our understanding of the software and make sure we're all on the same page. At this meeting, we collaborated to make our first rough draft diagram for how the software could be organized.

Upon finishing that first draft, we started to discuss potential architectural styles. Along with layered and interpreter styles, which will be discussed later on, we also considered a publish-subscribe style. Our reasoning for this was that it's possible that while running the game, the engine could announce events that the other components could respond to, such as cues for the audio and graphical components to perform their tasks. After some further discussion we realized that although this style could possibly fit, it doesn't make the most sense and isn't applicable to the rest of the system. After this meeting, we decided on a layered and interpreter style, but still had some doubts. Once we did further research into fully understanding the OSystem API component and the engine component, we were confident in our choice of architectural style.

4. Architecture

4.1 Architecture Style

The conceptual architecture of ScummVM follows a layered architecture style, which helps to organize the complex set of functionalities necessary to emulate and run classic point-and-click games. The biggest indicator that this is a layered architecture is the OSystem API, which acts as a layer that connects the operating system to the rest of the software. Each layer is responsible for certain tasks, from managing input/output to interpreting the actual game scripts, enabling a clean separation of concerns. This allows for scalability, enabling the addition of new game engines or features without disrupting the entire system. At the heart of ScummVM's layered architecture is a large interpreter system, which serves as a virtual machine to run platform-independent point-and-click game code. This subsystem interprets game scripts, such as those developed using SCUMM, AGI, or SCI engines; and translates them into real-time actions within the game. This allows them to be played across different operating systems without modification, preserving the original gameplay experience. In addition to the interpreter system, ScummVM's architecture includes modular components for key functions like audio, graphics, input handling, and game state management. These components interact with the game engine interpreter but can be updated or replaced independently - allowing for modular support and continued optimization of point-and-click games for a wide variety of platforms. The layered architecture style provides the flexibility and robustness needed to emulate a diverse set of games effectively.

4.2 System Overview

There are seven main components in the ScummVM architecture. The roles of these components and their interactions with each other will be the subject of this section.

Engines / SCI: The engines component is a collection of recreated adventure game engines that ScummVM uses to run adventure games. For the rest of the report, when referring to the game engines, we will specifically be talking about the SCI game engine. The game engine component is responsible for interpreting the data files of the game being run. The engine makes use of the audio component and graphics component to turn the interpreted data into something tangible that the user can see and hear.

Audio: The audio component of ScummVM is comprised of several different parts like drivers, decoder(s), and more. ScummVM includes built-in support for MIDI, allowing users to emulate or use a physical synth such as the Roland MT-32. This means that games played on modern systems are still able to sound exactly like they did on the systems that they were originally released for. The audio component is responsible for loading the appropriate audio asset as specified by the resource manager, mixing it, and initiating playback at the right time given events taking place in the game world

Graphics: The graphics component of ScummVM is responsible for managing the visual output presented to the player. This includes rendering both two-dimensional and three-dimensional game worlds, but also the GUI that the player will see. One key task of the graphics component is scaling. Since ScummVM runs games originally made for different displays, ScummVM includes a number of different methods for scaling the image up to larger sizes to better fit modern screens. There are even different rendering modes included, allowing ScummVM to emulate the looks of different systems; for a few older games which could be played on more than one system.
Input Handling: The input handling component is responsible for capturing user input. In the case of ScummVM here, that will come in the form of either mouse clicks, keyboard strokes, or input from joysticks or other game controllers. After the input is captured, it is passed to the engine for processing. It is important to consider here that users may be installing ScummVM on a multitude of systems, so the input handler may well have to process input from, say, a Wii controller and not just your regular mouse and keyboard.
Game State Management: The game state management component of ScummVM tracks the current state of the player's progress through the game. This includes several things such as variable values, events taking place both past and present, and more. ScummVM incorporates an autosave feature, triggering an automatic save of the current state of the game every five minutes, adjustable in a configuration file. Otherwise, the player is able to use a general function called the Global Main Menu to make a save and/or load a previous save from any point within a game. From the Global Main Menu, one is also able to resume their game, quit ScummVM entirely or just exit to the launcher, or even customize some gameplay settings in an options tab.
Launcher: The launcher is responsible for providing a graphical interface to the user, as well as managing engines. Within the launcher, users can load games from their system, remove games, launch games, and edit game-specific settings as well as ScummVM settings. It communicates with the platform abstraction component to access game files, as well as to load the game programs and corresponding engine into memory when being run.
OSystem API: The OSystem API component acts as a bridge between ScummVM and the platform's operating system. The API specifies various features a game can use. These could be things like displaying graphics on a screen, receiving input, etc. The API is then implemented for various different operating systems so that the games and engines are able to run on any platform that the OSystem API has been implemented for. The OSystem API interacts with the input handler when receiving input from the operating system, as well as with the launcher when it needs to access files and other operating system resources. It also receives calls from the audio and graphical components and passes on these calls to the operating system.

4.3 Development History

ScummVM started off as computer science student Ludvig Strigeus' efforts to create an engine capable of running *Monkey Island 2* on his Linux machine. He was soon joined by

Vincent Hamm, who was separately working on a SCUMM interpreter but abandoned his efforts after realizing that Strigeus was further ahead. After some time the implementation for both *Monkey Island II* and an additional title, *Indiana Jones and the Fate of Atlantis*, was finished.

After it was noted and reported on by news website Slashdot in early November 2001, there was an influx of interested developers. Work was able to progress in various parts of the project and the user base grew greatly. Even with all of this expanded interest and growth, the developers still feared legal reprisal, since they were replicating patented content. Nevertheless, they persisted. New games were added to the library of supported titles, and even entirely new engines to support them. Eventually, by mid-2002, both Strigeus and Hamm had wandered away from ScummVM to pursue other projects.

Despite the founders leaving, there were still around a dozen other active contributors at the time, so the project was still able to grow. It was roughly at this time that the entire project underwent a rewrite from C to C++. This was mainly an effort to improve readability and maintainability and to clean things up at the same time. A rewrite of the GUI soon followed which made its first appearance in release version 0.3.0, which was also notable as being the first release version to include a launcher as a frontend (previously the command line was used to launch games). The company Revolution Software even ended up donating the source code for two of its games to the ScummVM developers at the time.

More and more continued to be added to the project as time went on. The 14th of February 2009 saw the addition of the SCI engine (below) to ScummVM, first being included in release version 1.2.0. One particularly notable change was that the project, which initially only supported two-dimensional point-and-click games, grew to support even certain three-dimensional adventure game titles. Recently in 2021, ScummVM celebrated its 20th anniversary, a great testament to its longevity and continuously active development.

To this day, ScummVM continues to evolve. ScummVM is an open-source project, and so developers who are interested in the project continue to add support for more game engines so that a wider variety of games can be run, as well as support for more operating systems so that you can play these games on any sort of device.

4.4 SCI

4.4.1 Introduction

SCI refers to "Script Code Interpreter", and then later "Sierra's Creative Interpreter". SCI was created by Sierra as a new scripting language to write their adventure games in. It is used for running platform-independent, object-oriented code.

4.4.2 Architecture and Functionality

The architecture of SCI is an interpreter-style architecture. The main component of SCI is the PMachine. This is essentially a virtual CPU that executes the game being run. When the SCI interpreter is initialized, memory is allocated on the heap for the PMachine, as well as a pointer to the game object. When a script needs to be executed by the PMachine, an image/copy of it is also loaded into the heap. The PMachine has its own set of registers, as well as a stack, that is used to keep track of the current state of the program. The PMachine reads the instructions from the data files, which are files of machine code that have been compiled from Sierra's SCI compiler, and executes the instructions specified by these files to run the game.

4.3.3 History and Evolution

As Sierra continued to make games using SCI, they continued to work on SCI itself to increase its capabilities. The first version of SCI is referred to as SCI0. This engine allowed for 320x200 graphics, with 16 colours available. It also supported a music card-compatible soundtrack. For gameplay, it supported keyboard input, which was then checked with a text parser to perform actions in games. The engine supported a small amount of resources, which can be grouped into four main categories. VIEW, PIC, FONT, and CURSOR for graphics, SCRIPT and VOCAB (for parsing user input) for logic, SOUND and PATCH for sound, and TEXT for strings. The next version of SCI is SCI1. This was the biggest change for SCI. The engine now supports 256 colour graphics instead of the previously available 16. It also introduced a point-and-click interface, which allows them to create point-and-click style adventure games as opposed to the previous text-based games. They also expanded the resources to a much bigger list. They continued to create more versions, SCI 32, SCI 2 and SCI 3, however, the changes aren't significant and are essentially just higher resolution graphics and more resource types.

4.5 Control Flow

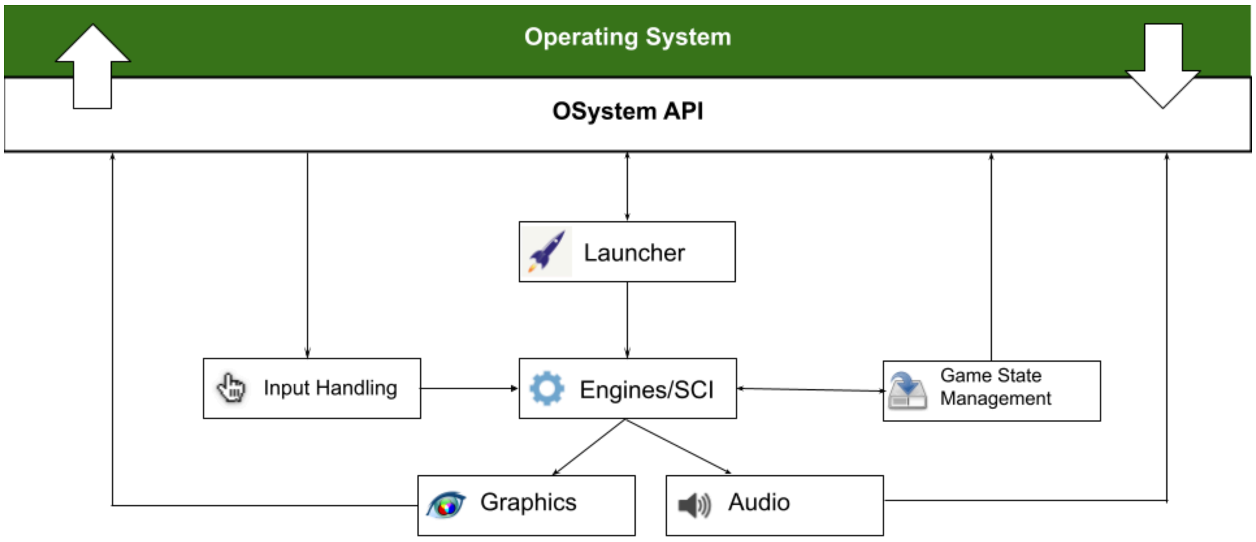

Figure 1: A dependency diagram of the ScumVM architecture.

One of the main important components of the architecture is the OSystsem API which sits between all the other components and the user's operating system. When a user wants to play a game, they first have to use the launcher which displays a GUI. Once the user selects the game the SCI engine starts up. The engine takes care of all the game logic. Whenever a user presses a button on their keyboard or moves their mouse the OSystem API sends a message through the Input Handling component, which then communicates with the engine. The Engine component also communicates with the Graphics and Audio components which then send data back to the OSystem API to allow for graphics and audio to be seen and heard by the user. When a user wants to save a game, the Game State Management component sends the save files through the OSystem API before the file is saved to the user's local storage.

4.6 Use Cases:

1. **Playing a Point-And-Click Game on ScummVM**

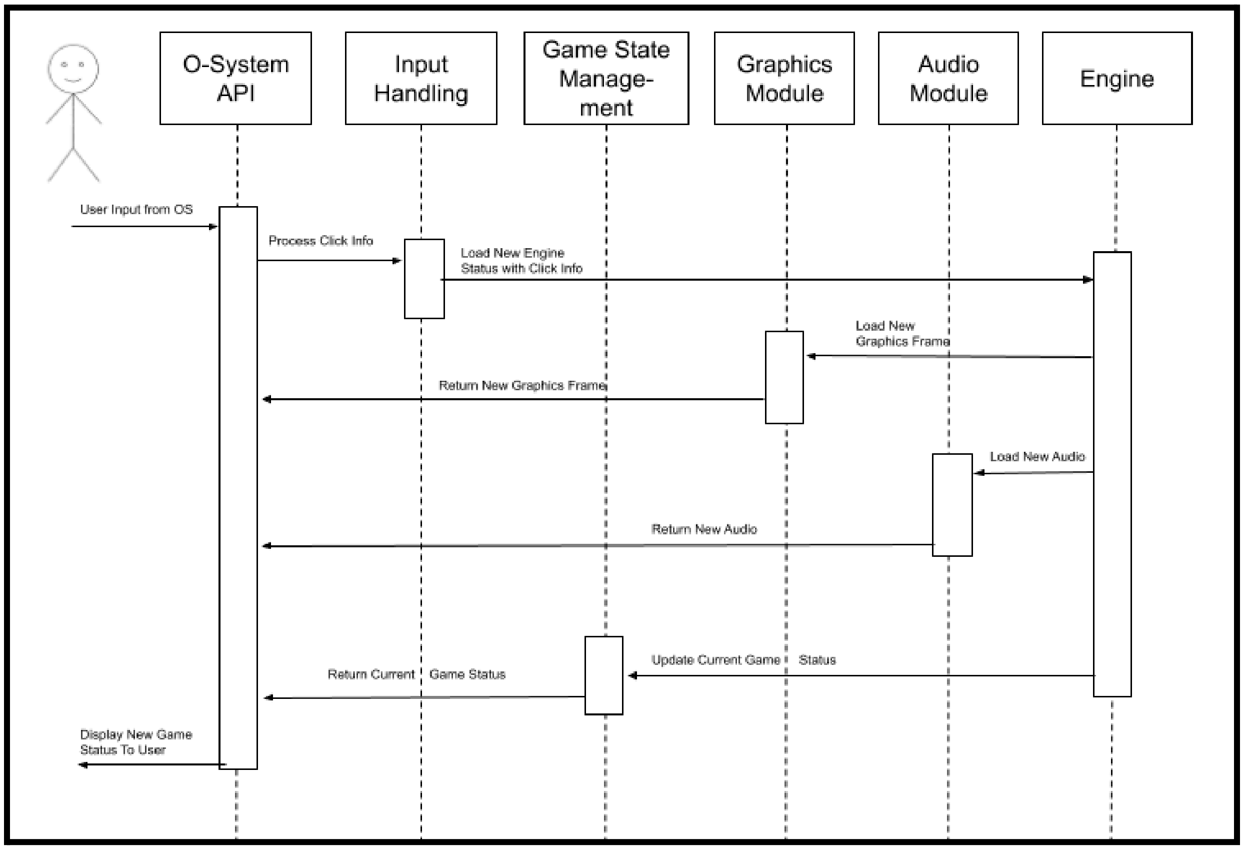

Figure 2: Use case 1: ScummVM loads the required components in order to update the status of a video game based on user input and displays the respective output

The first use case assumes that the user is currently playing a point-and-click game through ScummVM and gives some kind of input (mouse click) in order to play the game. As soon as the user provides input, it is sent to the O-System API in order to interpret the input into something the engine can understand. Firstly, the O-System API sends the input to the Input Handling module in order to process the input information, which then sends the processed input to the engine in order to load a new engine status based on the user's input. After the engine receives this new input info, it then goes on to update the various modules required to play the game. Firstly, the engine loads the graphics module, which returns the updated graphics based on the given user input to the O-System API in order to be converted into graphics that the user's operating system can understand. The engine then does the exact same for the audio module, converting the updated audio based on the given user input into audio that the user's operating system can also understand. Finally, the engine updates the current game status based on the given user input, and returns it to the O-System API in order for it to be loaded to the user. After the O-System API has all updated elements of the game loaded successfully as well as any save data, it now displays the updated game status of the game to the user, who continues to play the game.

2. **Launching a Game on ScummVM**

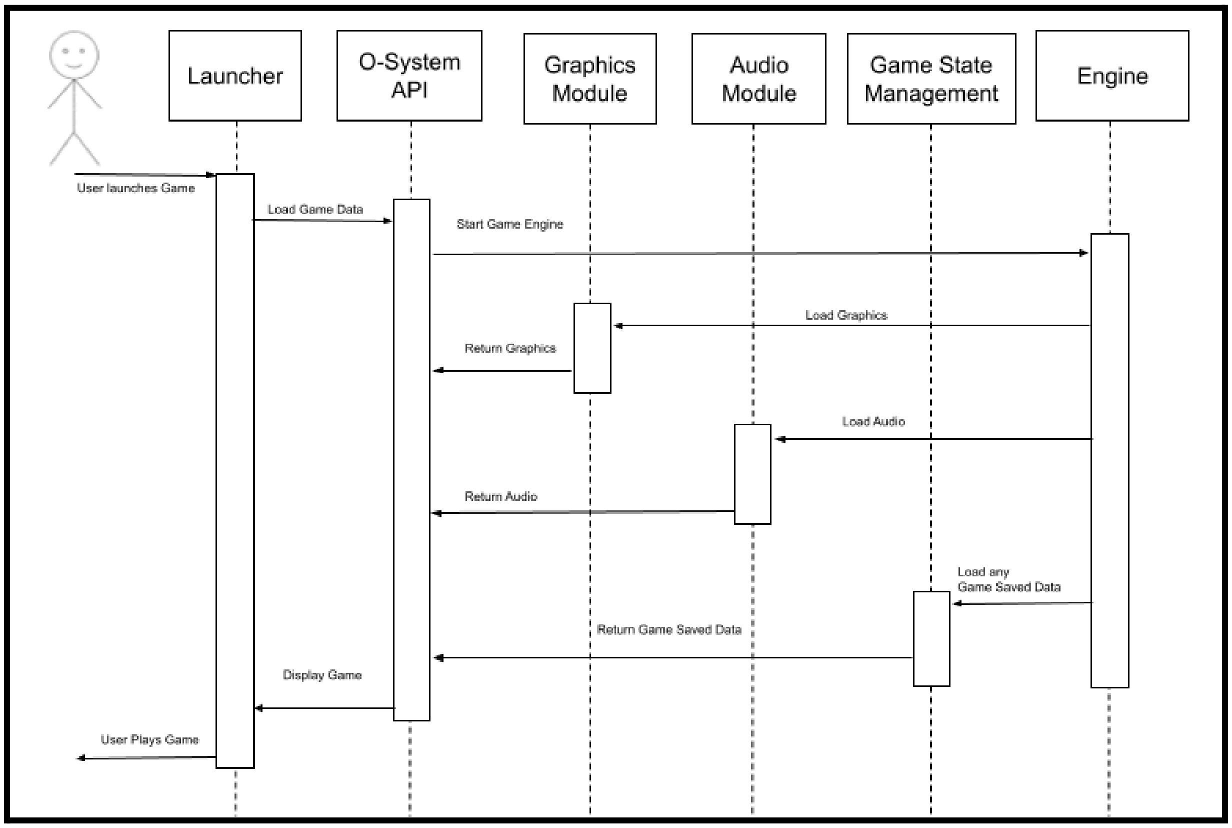

Figure 3: Use case 2: ScummVM loads the required components to successfully launch a point-and-click game

The second use case assumes that ScummVM is booting a point-and-click game from the initial ScummVM launcher. It starts off with the user choosing a game to launch through the launcher. After the user has selected a game to play, the launcher calls the O-System API to start the conversion of the engine of the selected game in order to run and play the game. The O-System API then calls the engine to start loading the various modules required to play the game. Firstly, it loads the graphics module, which returns the graphics to the O-System API in order to be converted into graphics that the user's operating system can understand. The engine then does the exact same for the audio module, converting the original audio into audio that the user's operating system can also understand. Finally, the engine loads any game-saved data that the user may have from previous game sessions and returns it to the O-System API for it to be loaded to the user. After the O-System API has all elements of the game loaded successfully as well as any save data, it now displays the elements of the game to the user, who continues to play the game.

5. Data Dictionary

API: Application Programming Interface. A piece of software which facilitates communication/functioning between two different computers or other pieces of software
SCI: A game engine created by Sierra
MIDI: Musical Instrument Digital Interface. Not sound itself, more comparable to digital sheet music
GUI: Graphical User Interface. Allows the user to interact with the system through graphical icons and other visual indicators as opposed to a text-only command line terminal interface

6. Naming Conventions

API: Application Programming Interface
CPU: Central Processing Unit
GUI: Graphical User Interface
MIDI: Musical Instrument Digital Interface
SCI: Script Code Interpreter
SCUMM: Script Creation Utility for Maniac Mansion
VM: Virtual Machine

7. Conclusions

Compared to when we were first introduced to ScummVM, we have a much better understanding of the system and its conceptual architecture from examining its documentation from when ScummVM was first created up until now. From analyzing the ScummVM system, we have come to the conclusion that it fits into a layered architectural style as it allows ScummVM. Furthermore, while layered style encompasses the entire system, at its core, we consider ScummVM as an interpreter architectural style as well. Looking at the architecture components, We identified 7 key modules which include the engines, audio, graphics, input handling, game state management, launcher, and OSystem API components. When referring to engines, we specifically wrote about the SCI engine. We found out that the SCI engine follows an interpreter style and has a core component that acts like a CPU called the PMachine; not only that, but we also discussed the SCI engine's history and evolution where it evolved to have more and more improvements to its resolution and resources.

Looking ahead, ScummVM has the potential for growth and development. The development team behind ScummVM still has many engines and games that they can look into interpreting. Also, with how ScummVM is designed to be able to play on any system, even with the release of new-gen consoles and their games, ScummVM won't easily be phased out since it will be able to run on those platforms. In essence, ScummVM is likely to continue thriving as the go-to software for those who seek enjoyment from playing classic adventure games for the foreseeable future.

8. Lessons Learned

During the analysis of the ScummVM software, we recognized the importance of clear and concise documentation as the complexity of the system made it difficult to understand. With there being various game engines, it became clear that documentation should not only be clear but also easy to find. ScummVM is an older project that has gone through many versions and updates, as well as having different contributors who add on top of pre-existing work. With the numerous contributions spanning over many years, it led to inconsistencies and a lack of a single unified source of information. The result was a system that was confusing to grasp and understanding the system required us to piece together information from multiple sources. In addition, ScummVM's need for constant updates, especially to maintain compatibility with new operating systems (like a new version of Windows), taught us the importance of ongoing maintenance and the need for adaptability in software design. Lastly, working on a project like this required effective teamwork and since we all didn't know each other from the start, we had to quickly learn how to collaborate given our different working styles. During the assignment as a whole, we learned the value of communication, organization and flexibility when tackling complex, long-standing software systems such as ScummVM and will apply these lessons to future deliverables.

9. References

[1] *jibbodahibbo, P. by, LittleToonCat, P. by, DreamMaster, P. by, djwillis, P. by, & scemino, P. by. (2024, October 3). Home. ScummVM. https://www.scummvm.org/*

[2] *SCI. (n.d.). ScummVM :: Wiki*. https://wiki.scummvm.org/index.php/SCI

[3] *Welcome to ScummVM! — ScummVM Documentation documentation*. (n.d.). https://docs.scummvm.org/en/v2.8.0/

[4] *Developer Central. (n.d.). ScummVM :: Wiki. https://wiki.scummvm.org/index.php?title=Developer_Central*

[5] *Programming a new game - ScummVM :: Forums. (2009, September 3). ScummVM Forums. https://forums.scummvm.org/viewtopic.php?t=7886*

[6] *Uurloon, M. (2015, December 1). Design of a point and click adventure game engine | Groebelsloot. https://www.groebelsloot.com/2015/12/01/design-of-a-point-and-click-adventure-game-engine/*

[7] *Deckhead. (2020, January 15). Game Engine Development: Engine parts. IndieGameDev. https://indiegamedev.net/2020/01/15/game-engine-development-for-the-hobby-developer-part-2-engine-parts/*

[8] *Folmer, E., Game Engineering Research Group, & University of Nevada, Reno. (2007). Component based game development – a solution to escalating costs and expanding deadlines? In H. W. Schmidt (Ed.), CBSE 2007: Vol. LNCS 4608 (pp. 66–73) [Journal-article]. Springer-Verlag Berlin Heidelberg. https://www.eelke.com/assets/pubs/cbgd.pdf*

[9] *Wikipedia contributors. (2024, October 4). ScummVM. Wikipedia. https://en.wikipedia.org/wiki/ScummVM*

[10] *Moss, R.* (2012, Jan 17). *Maniac Tentacle Mindbenders: How ScummVM's unpaid coders kept adventure gaming alive.* Ars Technica. https://arstechnica.com/gaming/2012/01/maniac-tentacle-mindbenders-of-atlantis-how-scummvm-kept-adventure-gaming-alive/