CISC 322
A1: Conceptual Architecture report
October 11, 2024

Conceptual Architecture of ScummVM

Group4:
Lorem Ipsum

Authors:
Dima Zaichenko (22bh17@queensu.ca)
Joshua Yarussky (21jy71@queensu.ca)
Roman Rodchenkov (21rr69@queensu.ca)
Mohammed Sarhat(19ms16@queensu.ca)
Charles Letellier(21cl106@queensu.ca)
Patrick Favret(21phf1@queensu.ca)

Table of Contents

Abstract

ScummVM is an open-source project designed to enable the play of classic adventure games on modern platforms by re-implementing original game engines. Initially developed in 2001 to run LucasArts' SCUMM-based games, ScummVM has expanded to support over 250 titles, including games from other companies. The system uses a combination of layered, interpreter, and object-oriented architectural styles, with 5 main key components like the Graphical User Interface (GUI), utilities, game engines, OSystem API, and backends. ScummVM's modular design ensures flexibility, portability, and maintainability across various platforms. The architecture abstracts platform-specific details, allowing vintage games to run on modern hardware without emulation. Though it lacks automation for adding new game engines, the system's open-source nature and strong community support have driven its evolution. ScummVM exemplifies how clear documentation, modularity, and persistence can ensure the long-term success of an open-source project.

Introduction and overview

Development of ScummVM began in September of 2001 when Swedish programmer Ludvig Strigeus created a tool to run Monkey Island 2, which used LucasArts' SCUMM engine, on his linux machine. Originally focused on supporting LucasArts' SCUMM-based games like Maniac Mansion and Monkey Island, the project grew quickly once Strigeus made it open-source, allowing for contributions from other developers.

By 2002, ScummVM was established as a virtual machine for running SCUMM games across multiple platforms (hence the name). Later, its scope expanded to support non-SCUMM games, allowing it to run titles from other companies such as Sierra's King's Quest. Since then, ScummVM has implemented many more game engines, enabling it to run over 250 titles. It has also been ported to numerous platforms, including desktops, mobile devices, and gaming consoles.Today, ScummVM is a community-driven, open-source project that continues to evolve by adding support for more engines and games, and allowing gamers to play retro games to their heart's content.

The purpose of this report is to explain and document the conceptual architecture of ScummVM. ScummVM stands for "Script Creation Utility for Maniac Mansion Virtual Machine" and it is an open-source project that is maintained, improved and added upon constantly by volunteers rather than a centralized team.

From the documentation, it is clear that it does not use any singular architectural style to achieve its purpose, but instead a “mish-mash” of the layered, interpreter, and object-oriented styles, though it is primarily using the layered style. There is a clear structure of “layers of abstraction”, from the lowermost hardware interaction layer (the “backend” that talks to the operating system and hardware), to the game engine abstraction layer (the “OSystem API”), and finally to the game engine itself. This is done so that the game engine can run on any machine (that ScummVM supports) without actually having to be re-implemented and tailored to the specific environment where it is running. The game engines themselves follow the philosophy of the interpreter style architecture, where they have access to the game files and act as a sort of state machine to run the game directly from its files and scripts. Finally, the object-oriented style is used throughout the system, as most of the project is done in C++, which is fundamentally an object-oriented language.

ScummVM emulates different game engines (note: ScummVM is *not* itself a game emulator) by replacing their original executables with its own implementations rewritten in C++. ScummVM acts as a virtual machine for different game engines, each of which interprets game logic, user input, and handles assets (e.g., images, sounds, and scripts). These engines are abstracted from the hardware layer by an interface that has platform-specific implementations for core operating system functionalities (see: OSystem API and Backend, under Architecture), making it possible for ScummVM to run on many different platforms without needing to rewrite game-specific logic for each operating system. This is especially important as many of the vintage games ScummVM is used for had to run on proprietary hardware (old consoles and such), with vast differences in the fundamental design of those systems, such as big vs little endian, processor word size (ie, 16 bit, 32 bit, etc), and more of such fundamental system protocols. It doesn't emulate the whole operating system or hardware from the past, but instead directly reads and processes the game data files, acting as a replacement for the original game executable.

The core of ScummVM's architecture is its modular design, where each supported game or family of games runs on a different engine, like SCUMM engine (which is where ScummVM gets its name from), or the SCI engine, used for Sierra’s adventure games such as King’s Quest or Space Quest. Each engine (of which there are many) is responsible for interpreting a game’s original script language, managing the state of the game, rendering graphics, playing sound, and handling input, the end result being the game “just working” on a modern device. ScummVM interprets the original data files (e.g., scripts, graphics, audio files) from the supported games. A feature which was not present in many of these older games, saving anywhere, has

been made possible by ScummVM, due to the nature of how it runs the games (as a state machine), it allows for saving anywhere.

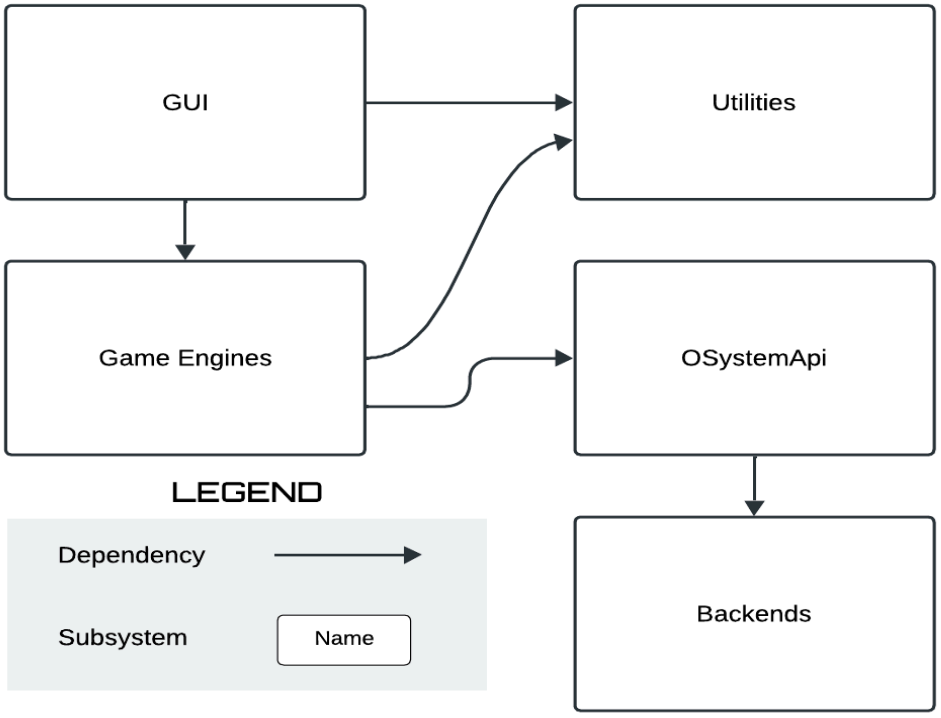
This report goes into further detail about the architecture, and how the system's various parts interact, in the Architecture section. There are sequence diagrams that demonstrate conceptual use cases and how the system performs them in the Use Cases section. For definitions and clarification of various acronyms and technical terms, see the Data Dictionary and Naming Conventions sections.

Ultimately, ScummVM is an open source project that allows for the playing of vintage games on modern hardware directly from the game files, by providing implementation of the necessary game engines that can interact with the operating system through an abstract interface.

Derivation Process

To Derive the conceptual architecture of scummVM as shown in figure 1(below), we began by scouring sources of information for clues as to the conceptual architecture of ScummVM. When we first went on the developer wikipedia it listed the 5 main components that make up the system, from this we are able to piece together the architecture of scummVM. Using these 5 components we went into figuring out how they are connected to and communicate with each other by looking at the source code in the github repository, this helped us get a picture of the architecture style. We figure out that the code is built around 3 styles, Layered Architecture, Interpreter and Object-Oriented Architecture. The overall conceptual architecture is layered with 3 tiers, the Top Tier consists of GUI and Utilities. The middle tier is the game engine and OSystemAPI and the 3rd tier is the Backends. And inside each of the components they are built around interpreter style allowing modifiability and object-oriented style also allowing modifiability and scalability. This is how we followed a derivation process to create the box-and-Arrow diagram shown in figure 1. With how well scummVM development is, it's hard to come up with alternatives to how the architecture could look, leaving us with no doubts as to this design layout.

Architecture



System Architecture Box-and-Arrows Diagram (figure 1)

ScummVM is designed as a modular system that facilitates the execution of classic adventure games across a variety of modern platforms. Its design emphasizes flexibility, maintainability, and portability, achieved through a combination of architectural styles: **Layered Architecture**, **Interpreter Style**, and **Object-Oriented Architecture**. The system is structured into five primary components: the **Graphical User Interface (GUI)**, **Game Engines**, the **OSystem API**, **Backends**, and **Utilities**. Each component has distinct responsibilities and interacts with others through well-defined interfaces, ensuring a cohesive and extensible system.

Graphical User Interface (GUI)

The GUI serves as the user's gateway to ScummVM, providing an intuitive interface for game selection, configuration, and management of saved games. Designed to focus on usability and accessibility, it supports multiple languages and

themes to cater to a wider user base. The GUI is platform-independent, relying on the OSystem API for rendering graphics and handling user input, which allows it to maintain consistent functionality across different modern operating systems without modification.

Interaction between the GUI and other components is central to ScummVM's operation. When a user selects a game to play, the GUI communicates this choice to the appropriate Game Engine, passing along any user-defined settings. It utilizes the Utilities component for tasks like reading configuration files and managing resources. By abstracting platform-specific details through the OSystem API, the GUI remains adaptable to future changes, promoting software evolution and simplifying the process of updating or enhancing the user interface without impacting the underlying game execution logic.

Game Engines

ScummVM functions by using rewritten game engines, each built to understand and run games that were originally made with specific operating systems or scripting systems. Instead of needing the original executable files, these engines read the game's data files and run the scripts, essentially recreating the gaming experience to work on other operating systems. This design follows an interpreter architectural style, allowing ScummVM to support a variety of different games.

The game engines work closely with the OSystem API component to perform system-level operations like rendering graphics, playing audio, and handling input from the player. They also use various utilities for things like managing files, parsing data, and dealing with memory. The whole system is modular, so you can add new engines to support more games without messing up what is already there. This makes it easier to test since you can test each engine on its own. Plus, the system is flexible and can easily be updated to handle new games or features as they come along.

OSystem API

The OSystem API is the interface that defines what available features a game can use, like drawing on the screen or receiving keyboard and mouse events. The OSystem API shields game engines and GUI code from the actual platform the software is running on, acting as a layer of abstraction between the Game Engines and whichever operating system (and hardware) is being used - to understand ScummVM, you can ignore everything in the backends code as you don't need to know how the OSystem API is actually implemented. This design follows the layered architecture style, as it follows the philosophy of "Each layer provides service to the layer above it and serves as a client to the layer below it", in this case providing service to the "Game

Engine” layer and acting as a client to the “Backends” layer, where this API is actually implemented. By delegating platform-specific operations to the Backends, the OSystem API simplifies the process of porting ScummVM to new platforms. Developers need only implement the OSystem API for a new platform's Backend, leaving the Game Engines and GUI untouched. The OSystem API's design ensures that future changes in hardware or operating systems can be made with minimal disruption to the rest of the system.

Backends

The backend component is responsible for the concrete implementation of the OSystem API across the various platforms in the system. Each component inside the Backend subsystem deals with specifics of interacting with the OS and the corresponding hardware. This component is crucial for dealing with the performance non-functional requirement, since the backend deals with window management, input processing, graphic rendering, and audio processing. An important non-functional requirement for a platform like ScummVM is performance, this component is where this NFR is most dealt with since it must efficiently manage system resources and responsively handle interactions between the user and the game. The Backend is built around a modular nature, which allows the system to work on a large range of platforms, it also future-proofs the systems since the backend can be developed to extend the compatibility. Creating an effective scalable system, ensuring each specific environment is optimized. Also by encapsulating the platform-specific code within the backends, the system stays portable.

Utilities

The utilities form a critical part of the ScummVM architecture. They are composed of helper functions, libraries, modules, and utility classes that provide common functionality across different systems.These include common data structures, file I/O routines, audio and video codecs, and other tools. Since utilities are designed independently of any operating system they allow for code reusability and consistency. Reusing and centralizing this code, leads to a simpler and more maintainable architecture that is less prone to error.

ScummVM’s utilities achieve platform independence by relying on the OSystem API to make calls to the user’s system when needed. This layer of abstraction handles the platform specifics so that utilities can run on multiple platforms without modification. Platform independence is an essential aspect of ScummVM because it needs to be able to work on a wide range of devices and operating systems. ScummVM’s modular

design lends itself to increased testability and evolvability; any bug fixes automatically benefit all components that rely on utilities. This design also allows for the creation of new features without impeding the existing architecture.

Interactions and Control Flow

When the application starts, the GUI initializes and presents the user with options to configure and select games. Upon selection from a user, the GUI invokes the Game Engine. This is mainly done to pass along any configurations that might be deemed necessary. From here, the Game Engine begins interpreting the game's scripts and utilizes the Utilities, with the aim of completing common tasks such as data management, and file access. Furthermore, the Game Engine makes calls to the OSystem API for operations like rendering graphics or processing a given input. The OSystem API, in turn, delegates these requests to the backend, which executes the platform-specific code to interact with the hardware.

Architectural Styles and Future Support

ScummVM's architecture makes use of several architectural styles to meet its goals effectively. The Layered Architecture provides a clear separation of tools, making the system more manageable by employing a hierarchical system. By dividing responsibilities among layers, GUI, Game Engines, OSystem API, Backends, and Utilities, the system supports independent development and testing of each component. The Interpreter Style is central to the Game Engines, enabling ScummVM to execute game scripts and support a wide variety of games by rewriting their executables. This approach enhances the system's flexibility and allows for the preservation of games across different, more modern platforms. An Object-Oriented Architecture style supports the entire system, promoting encapsulation, inheritance, and reusability. This makes it easier to integrate new features, support additional games, and maintain the overall system. By using object-oriented principles, ScummVM is able to make use of some of the benefits of the style, ensuring that its components are modular and that interfaces between them are well-defined.

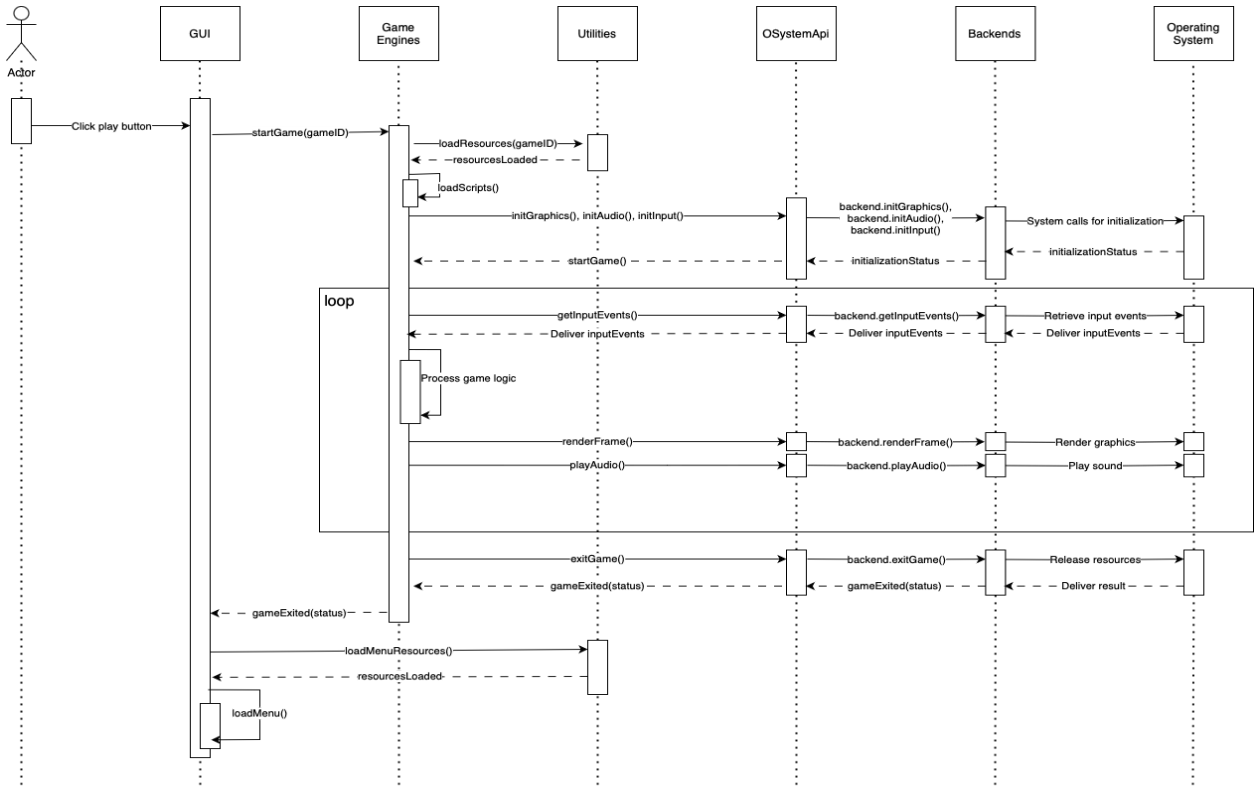
This combination of architectural styles supports ScummVM's evolvability and scalability. New Game Engines can be added to support more games, new backends can be developed for emerging platforms, and the GUI can be updated to enhance the user experience, all without significant restructuring of the existing codebase, making it perfect for future updates.

Concurrency

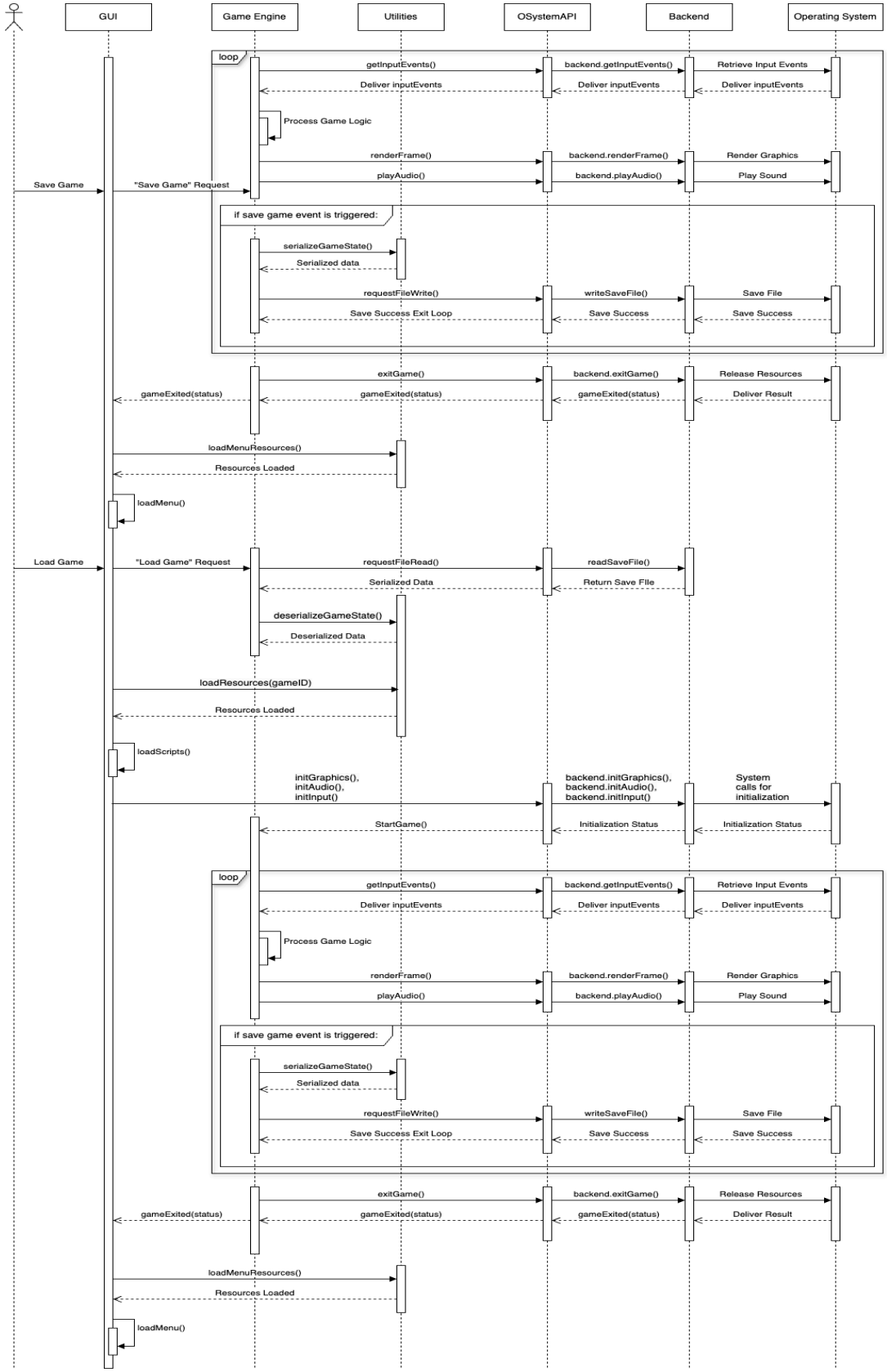
There is minimal concurrency utilized in scummVM as it is run on a single thread, since the majority of games run on scummVM engine are older they don't need multithreading so they benefit from the more simple single threading, however there is automatic cooperative threading, which helps the single thread processing mimic concurrency. This works by having multiple scripts queued to be processed and when the current script is stopped either by spawning a child thread that needs to be processed before it, it froze or has a delay built in another thread will take over, this leads to seamless process transitions making it seem like there concurrency, but in reality everything is on 1 thread.

Use cases

Sequence Diagram for starting game use case:



Sequence Diagram for saving and loading games:



Limitations and Lessons Learned

ScummVM lacks inherent scalability. While it supports a wide variety of classic games to be run on modern OS's, it suffers from a lack of automation, as each game must have its original engine re-worked by an individual developer contributing to the project, a process which can take weeks if not months. To rectify this limitation, the ScummVM website provides a compatibility list indicating which games are fully supported, partially supported, or unsupported. Furthermore, whether by choice or not, ScummVM seems to only provide support for classic games- those wishing to port modern games from a more modern OS would have to go looking elsewhere, leading to a smaller audience. Though it can be argued that the point of ScummVM was never to pander to a larger player base, as very few people are actually interested in the antique games that ScummVM provides, they can be seen as fulfilling this 'niche', and would therefore be the go-to provider for their services.

The lesson learned from ScummVM, a relatively successful open-source project, would be to have patience and persistence when developing software, as while not everything can be automated, it is important to approach problems with a sort of meticulousness so as not to provide a lesser, non-functional product. It is also useful to have a list of compatibility if developing a similar project, as users going into a game with no prior knowledge of its compatibility with a more modern system may be confused or disappointed when faced with issues and bugs. Furthermore, ScummVM is well-documented, making it easier for more developers to join in the open-source contribution; clear and concise documentation is a necessity for achieving this. A large reason for ScummVM's success may also come from its modular nature, making it easier to expand upon the software with new games, thereby increasing the project's lifespan.

Conclusion

In conclusion, ScummVM is a vast project spanning decades of developer work and history. The architectural styles of layered, interpreter, and object-oriented programming allow for its flexibility and maintainability. ScummVM has preserved gaming history and provides an excellent example of open-source collaboration.

Data Dictionary

Layered Architecture: Layered architecture is a software design pattern that organizes a system into a series of distinct layers, each serving a specific purpose. This structure allows software components to interact in a hierarchical or tiered manner, where each layer communicates primarily with the layer directly above or below it. This design promotes the separation of concerns, enhancing maintainability, scalability, and testability of the software.

Interpreter architecture: Interpreter architecture is a design pattern frequently employed in the implementation of programming languages, especially scripting languages. It enables a program to execute instructions written in a high-level language by interpreting them directly instead of compiling them into machine code. It also allows a program to become modifiable, since you can use the interpreter to add extra code and combine it with the base code. Typically, this architecture comprises two primary components: the parser and the interpreter.

Object-Oriented Architecture: Object-Oriented Architecture is a design pattern based on the concept of objects. These objects represent both data, and functions that coincide with manipulating the data. It enables a focus on organizing a system as a collection of interacting objects, which makes the system modular, and maintainable. The foundational principles of Object-Oriented Architecture include encapsulation, inheritance, polymorphism, and abstraction.

Naming Conventions

SCUMMVM: The abbreviation for Script Creation Utility for Maniac Mansion Virtual Machine, which is the underlying virtual machine that allows ScummVM to run classic adventure games across various platforms.

GUI: The abbreviation for Graphical User Interface, a component that interacts with the user. This allows them to select games, configure settings, and manage saved data.

API: The abbreviation for Application Programming Interface. This allows ScummVM components, such as the OSystem API, to communicate and interact with different operating systems.

OSystem API: The abbreviation for Operating System API which represents a layer within the system architecture that deals with platform-specific details. This allows game engines to run on multiple platforms without issues.

NFR: The abbreviation for Non-Functional Requirement, which refers to system attributes like performance and scalability, amongst others, that the backend components must efficiently manage to ensure responsive interactions between the user and the game.

References

Welcome to ScummVM! — ScummVM Documentation documentation, https://docs.scummvm.org/. Accessed 5 October 2024.

ScummVm Wiki, 18 June 2023, https://wiki.scummvm.org. Accessed 5 October 2024.

ScummVM Forum, https://forums.scummvm.org/viewtopic.php?t=7886.

“ScummVM.” *Wikipedia*, https://en.wikipedia.org/wiki/ScummVM. Accessed 5 October 2024.

“SCUMM/Virtual Machine.” *ScummVM*, wiki.scummvm.org/index.php?title=SCUMM%2FVirtual_Machine&mobileaction=toggle_view_desktop. Accessed 9 Oct. 2024.

“Developer Central.” *ScummVM*, wiki.scummvm.org/index.php/Developer_Central. Accessed 5 Oct. 2024.