CISC 322/326 Assignment 1
Conceptual Architecture of ScummVM

Oct. 11, 2024

Group name: '; DROP TABLE groups; –

Xavier Awadalla - 21xpa@queensu.ca

Christopher Gil - 21cagi@queensu.ca

Aaron Rivest - 21alr18@queensu.ca

Craig Tylman - 21crt6@queensu.ca

Sam Tylman - 21sdt6@queensu.ca

Felix Xing - 21fx2@queensu.ca

Abstract

This report provides a comprehensive analysis of the software architecture of ScummVM, an open-source software designed to run classic point-and-click adventure games across multiple platforms. We will discuss the general architecture of ScummVM; the many software components which comprise the application and the interactions/(i forget the word they used) between them, focussing particularly on the Engine, the Graphics, Audio and LAN (Local Area Network) components, the GUI (Graphical User Interface), OS (Operation System) Abstraction and Storage Management; how the application handles concurrency and multithreading on a user's operating system; the division of responsibilities of contributors to the ScummVM project and how their work is organized and pulled into the main codebase; some attempts at illustrating use cases of ScummVM via sequence diagrams; an overview of the system's history and evolution as well as how development will continue into the future. We will also briefly give our concluding remarks from the findings we've gathered about ScummVM, and give a set of references that were used in obtaining said findings.

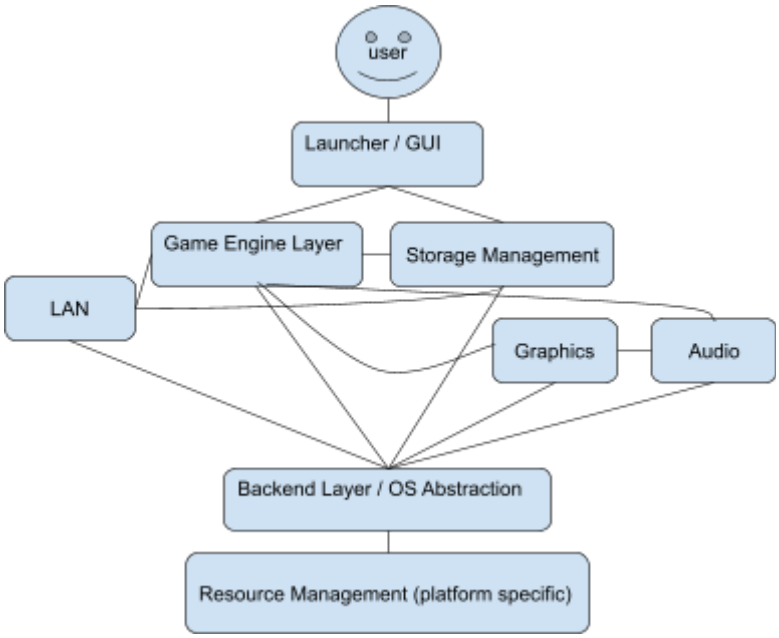
Introduction and Overview to ScummVM:

ScummVM (Script Creation Utility for Maniac Mansion Virtual Machine) is an open-source program that was launched in 2001 to run classic point-and-click adventure games that were originally developed for other platforms. This includes games such as LucasArts' SCUMM based games, Sierra AGI (Adventure Game Interpreter) and SCI (Sierra Creative Interpreter) games and many more. ScummVM replaces the executable files shipped with these games to allow users to experience the games on modern systems that otherwise would not be able to support them. In addition to this, ScummVM has many other handy and convenient features including:

- Saving game progress (manually or automatically)
- LAN integration
- Cloud storage to save, load or download game files from. Supported services include Dropbox, OneDrive, Google Drive and Box.
- Automatically checking for and downloading updates
- A UI to easily search for and launch games that the user has downloaded.
- A wide variety of game settings to adjust like graphics, key binds and audio.
- Support for Text to speech
- GUI settings
- A support page to inform users of useful information like important keybinds, how to set up cloud services and where to find and download supported games.

The goal of this document is to explain the conceptual architecture of ScummVM with explanations of the system's components and diagrams of said components interactions and use cases. Additionally, this document will discuss the division of responsibilities among participating developers and the evolution of the system.

ScummVM Conceptual Architecture:



This box-and-arrow diagram represents the conceptual architecture of ScummVM, showcasing key components and their interactions. The **User** interacts directly with the **Launcher**, which serves as the main interface for managing games and accessing low-level components such as the **Game Engine Layer**, **Storage Management**, and **Backend Layer**. This design allows users to navigate the system without needing in-depth knowledge of the underlying software architecture.

The Components

The Engine component

In ScummVM, “An engine” is not a component of the software, rather, multiple engines combined to form this component. Engines are the highest layer of ScummVM, they are responsible for directly reading the game binary (provided outside of ScummVM) and displaying dialogs, reading scripts and media files to lower layers for processing. Each engine is designed with an interpreter style so it can easily translate what mentioned above within a game binary in a logical way. Typically, a 3D engine needs to translate graphics draw calls from game binary to OpenGL (an industrial standard application programming interface for computer graphics) draw calls with or without shader support, but a software 3D renderer is also supported.

The simplest game engine is Plumber. Plumber is designed to run *Plumbers Don't Wear Ties*, originally published on 3DO Interactive Multiplayer. This game engine contains 3 main parts: a detector of game files for a given directory, a console and the actual game engine. *Plumbers Don't Wear Ties* is a visual novel/dating simulator, therefore the plumber

engine will extract image files, audios and dialogue out of game files and read input from the user's cursor for the most of the time rather than rendering geometries heavily.

The Graphics component

ScummVM usually supports at least 2 graphics backends on different platforms, most commonly being: a CPU software backend and OpenGL GPU (Graphics Processing Unit) backend. Due to the limitation of OpenGL itself, all draw calls must be made within the main thread. However, that allows engines to draw OpenGL calls directly without any translation layer if they are called within the main thread. It is also possible to change the renderer of games, this will emulate the behavior of a specific hardware to ensure compatibility. Most games that will be played on ScummVM are perhaps 2D games, and ScummVM provides shader support to change the look of the game and upscaling filter support for high resolution display. Just like 2D graphics backends, ScummVM also provides a CPU software backend and an OpenGL (with or without shaders) backend for 3D games with the support of anti-aliasing.

Software Renderer: The CPU backend relies on a well-known graphics library SDL. Just like ScummVM, SDL is a platform independent library, hence it has native implementation of different platforms and provides platform-agnostic APIs. For example, SDL uses GDI (Windows Graphic Device Interface) for software rendering and handles input events using Win32 APIs.

OpenGL Renderer: The OpenGL renderer on the other hand is more complicated than the Software Renderer. SDL is a versatile library, it not only provides software rendering but also provides hardware-accelerated rendering using APIs like DXD3D (DirectX 3D), OpenGL or Vulkan (by later version of SDL2). In ScummVM, only the OpenGL backend of SDL is used, since it is the most widely used and supported industrial standard API.

The Audio Component

ScummVM supports audio playback and synthesization through sound hardware, emulation of hardware or software synthesizer. Although there is support for auto detecting music devices for a given game, it is also possible to select global or game specific devices for playback. By default, ScummVM will use MT-32 or General MIDI device if not specified. Typically the game engine will pass MIDI information to the audio layer/mixer for playback, it is also possible to use OSystem APIs for playback as well.

General MIDI device: If not specified and ScummVM is built with FluidSynth support, FluidSynth will be the default MIDI (Musical Instrument Digital Interface) sound player. FluidSynth requires soundfonts to synthesize music, which needs to be provided externally. If FluidSynth is not available, ScummVM will use General MIDI devices available reported by the operating system.

MT-32: MT-32 was originally an external hardware for audio synthesization, but any devices(or emulated devices) that are compatible with MT-32 can be used with ScummVM for playback. Just like FluidSynth, MT-32 also uses MIDI protocol.

The LAN component

ScummVM can act as a local web server as an alternative to cloud storage. A local webserver using WIFI sharing feature based on SDL_net (an optional extension library of SDL) can be enabled. When ScummVM is acting as a local web server, it provides 2 file manager implementations: "/files" and "/filesAJAX". When ScummVM is acting as a web server, it will be running as a server within client-server mode, while other computers that are connected to it act as clients.

The GUI Component

The ScummVM Graphical User Interface (GUI), also referred to as the “Launcher”, opens whenever a user starts ScummVM. It acts as the main point of interaction for users to manage, configure, and launch games. With its simple design and yellow & orange color palette, the GUI is user-friendly and highly functional. However, users also have the ability to launch ScummVM directly into a game without opening the Launcher first. This can be done by using the command line, and providing ScummVM a target game ID or path.

Visual Layout

- **Search Box:** Allows users to filter games in real-time.
- **Grouping Menu:** Offers sorting options (e.g. Engine Type, Platform, Year).
- **Game List:** Displays added games in either a List or Grid view.
- **Side Buttons:**
    - **Start:** Launches a selected game.
    - **Load:** Opens a save file.
    - **Add Game:** Opens a file browser for game addition (with Mass Add).
    - **Game Options:** Adjusts game-specific settings.
    - **Global Options:** Modifies technical settings like Graphics, Audio, LAN.
    - **About:** Displays build information.
    - **Quit:** Closes ScummVM.
- **Global Main Menu:** Found only while playing a game. Can be opened by pressing CTRL+F5. It has many of the same buttons from the Launcher, with the addition of Resume (closes GMM), Save (saves the game state), Help (displays keyboard shortcuts), and Return to Launcher (quits game and returns to the Launcher).

The GUI Architecture

ScummVM's GUI operates through a **floating layout system of widgets**, built in C++. The **event-driven architecture** processes user actions via an event loop, updating the interface dynamically. Widgets are derived from a base ‘Widget’ class and include:

- **Graphics Widget:** Used to display images
- **Button Widget:** Widely used in the GUI for buttons
- **Static Text Widget:** Displays simple text
- **Container Widget:** Blank container to hold other widgets, such as the games list Widget, which contains widgets for each game.
- **Picture Button Widget:** A button with an embedded image

Additionally, the GUI has a Theme Engine that can interpret .STX theme files, which are a basic subset of XML that contain information for graphical assets such as cursors, button icons, widget layouts, and render info (e.g. colours, fonts, padding, etc.). Some **common events** that affect the interface are:

- **Button Presses**: Clicking a Side Menu button triggers a callback function to perform its task (e.g., starting a game or opening a window). For adding games, the GUI may call the OS's File Explorer, though ScummVM has a built-in file explorer that can be set as default in Global Options.

- **Search and Filter Updates**: Typing in the search bar or changing the sort filter updates the game list in real-time by triggering an internal sorting or search function without needing to reload the entire GUI.

- **Dynamic Resizing**: The GUI adjusts automatically to window resizing, ensuring all elements remain accessible and responsive, avoiding any lag or freezing.

- **Keyboard Shortcuts**: ScummVM includes various shortcuts for quick access to frequently used settings and actions.

The GUI serves as a key intermediary between users and ScummVM's core components, enabling easy adjustment of low-level settings without deep knowledge of its underlying architecture. Key functions include:

- **Event Handling and Input Processing:** The GUI processes user inputs (like launching games or adjusting settings) and forwards them to relevant components, such as the Game Engine for starting games or the OS's sound controller for volume adjustments.
- **Settings and Configurations:** After applying settings, the Launcher saves them in a configuration file (e.g., %APPDATA%\ScummVM\scummvm.ini on Windows).
- **File System Access:** The GUI simplifies file access through its built-in or OS file explorer, allowing users to easily navigate directories and locate game files without manually entering paths.

OS Abstraction (Operating System Abstraction)

One of ScummVM's standout features is its ability to run across multiple platforms without significant code rewrites. This is achieved through **OS abstraction layers**, which

separate game engines from the Launcher, enabling easy porting to environments like Windows, macOS, Linux, Android, and game consoles.

Built on a virtual machine that handles input/output, memory management, and other low-level operations, ScummVM allows game engines to focus on game logic while maintaining a consistent Launcher interface across platforms.

ScummVM uses platform-specific backends to manage components like Graphics, Audio, File Systems, and I/O, adapting to each platform's OS and hardware. These backends are modular, allowing game engines to function as plugins without altering ScummVM's core architecture. Additionally, new backends can be developed using the **OSystem** class, enabling support for more platforms. Backends can be tested using the "testbed" engine to ensure proper functionality.

The Storage Management Component

Obviously, many aspects of ScummVM cannot just exist in memory during runtime. Game files, configuration files and obviously the application/binary files exist in permanent storage (e.g. hard drive, SSD/Solid State Drive and cloud services). We will discuss how ScummVM manages said files below.

The storage of game files themselves is very straightforward. The user may place game files into a directory wherever they choose on their system, provided the application is given permission to access it. We will address the process for adding said games later, but for now it suffices to say that they are recognized by ScummVM upon selecting the path to the folder on the user's system, via the main menu that appears on starting the application.

Other files are managed by the application a little more directly. The user may store game save files, themes, icons and any "extras" that may or may not be required to run the games themselves are stored in paths specified by the user in the settings menu.

The system also assigns default paths to more "necessary" files like saves and icons, whereas the extras and themes are not found unless a folder is specified by the user. Note that Themes and Extras are not created by the application like saves, but rather loose files placed into the path by the user not unlike the game files. The main application configuration and console log file paths are also listed, but these are not changeable - they are set by the application upon a fresh install.

For users who either lack the storage space on their device or wish to share games and save files between devices (e.g. between an Android phone and a Windows PC), ScummVM also offers remote storage via "Active Storage" under the Global Options menu. The user gives ScummVM limited access to their drive on an external cloud storage provider (out of Dropbox, OneDrive, Google Drive and Box). After the integration is set up, a folder will then be created on the drive, called "ScummVM" and, once enabled, will

allow the user to sync their local files with that of the ScummVM folder on their cloud storage provider's drive.

Platform Backends

In ScummVM, backend entry points serve as a crucial integration component between the ScummVM engine and the hardware or platform it runs on. Each supported platform or device typically has its own backend that manages platform-specific tasks such as input, output, and graphics rendering.

The backend entry points are primarily handled through a component called "OSystem". OSystem is an interface which abstracts the platform-dependent operations and defines methods for input handling, graphics, sound, and timers. OSystem serves as the bridge between ScummVM's engine-independent core and the specific backend implementations required for different operating systems and hardware environments. For each supported platform, ScummVM relies on specific implementations of the OSystem interface, typically found in platform-specific files.

One such example of this is SDL. Simple DirectMedia Layer (abbreviated as SDL) is a very popular multimedia library which provides low-level access to hardware components (namely graphics, audio and input devices). For devices with SDL support, ScummVM provides a dedicated SDL backend. This ensures easy compatibility for Windows, Linux, macOS and some versions of Android and iOS.

Other platforms, such as iOS 7 or the Nintendo 64, may have custom backends with their own OSystem implementations, optimized to handle their unique hardware constraints. These platform-specific files handle everything from screen resolution adjustments to input device mapping. They adapt ScummVM's core functionalities to work seamlessly on different systems. These files ensure that the initialization and teardown processes for ScummVM are executed smoothly, and they manage platform-specific quirks like touch input on mobile devices or rendering pipelines on consoles.

Misc. Integrations

Alongside cloud storage, ScummVM has a handful of other external integrations which may be of use to those running the software:

Discord Presence: ScummVM has full support for the instant messaging platform Discord's rich presence feature. This feature allows developers to install a very lightweight SDK that automatically connects to the user's Discord client. From there, a custom 'status' can be displayed on the user's Discord profile, displaying exactly what they are doing in the connected application. In particular, ScummVM uses this feature to display the name of the game being emulated, as well as its logo and how long it has been running for.

Automatic Updates: ScummVM also features automatic update checking, by using the Sparkle framework. At a user-specified interval ranging from daily to monthly, the application automatically makes a server request that cross-checks the running version with the latest one available. If a newer version is found, the release notes will be displayed in a pop-up and the user will be given the option to quickly download it. Updates can also be manually checked for via the global options menu.

Accessibility Tools: ScummVM offers a variety of accessibility options, which can greatly improve the gaming experience for users with potential impairments. One such feature is a text-to-speech option, which connects to the screen reader provided by most operating systems so that UI components can be automatically narrated. Furthermore, there are many settings that can be changed on a per-game basis such as custom key bindings, subtitles with customizable speed, and adjustable volumes for music, SFX, and speech.

Concurrency

The **ScummVM launcher** has **minimal concurrency**, primarily following a single-threaded, event-driven architecture. It processes inputs one at a time through an event loop, which gives the appearance of real-time responsiveness but lacks parallel processing. However, background tasks like scanning directories or loading resources may be offloaded to separate threads to prevent the interface from freezing.

For **games running within ScummVM**, the concurrency model largely depends on the original game engine being emulated. Many older game engines were designed to be single-threaded with little to no concurrency, as they targeted simpler hardware. An example is the SCI engine, which is a p-machine style virtual machine (VM) designed to execute platform-independent, object-oriented code for 16-bit little-endian computers.

The original **SCUMM engine** from 1987, however, features **cooperative multitasking**, where tasks are divided and switched between in a way that simulates multitasking without parallelism. SCUMM supports up to **25 cooperative threads**, each executing instructions until suspending and passing control to the next thread. This system allows multiple game elements (e.g., animations or NPC behaviors) to progress independently, but only one thread executes at a time.

SCUMM Virtual Machine and Thread Management

The SCUMM VM uses **automatic cooperative threading**, where the flow of execution is transferred between different scripts based on their states. Each script gets the opportunity to run once per frame, unless it is delayed, frozen, or otherwise paused. SCUMM threads can exist in the following states:

- **RUNNING**: Actively executing.
- **PENDED**: Paused because it spawned another thread. The parent thread resumes when the child thread finishes.
- **DELAYED**: Paused and waiting for a timer to expire.
- **FROZEN**: Suspended until explicitly thawed.

Threads in SCUMM can be nested up to 15 times, and the stack is shared across all threads. In version 6 of SCUMM, it becomes the responsibility of the script to manage the stack carefully, especially when descheduling, as leaving items on the stack can cause issues. If state needs to be preserved between frames, it should be saved in local variables.

This threading model ensures a smoother gameplay experience by allowing tasks like animations, music, and NPC movements to progress while maintaining the original game behavior. The cooperative model also ensures that no thread is preemptively interrupted, preserving the flow of gameplay, as the control always returns to the launcher for handling input and output.

In summary, while the **ScummVM launcher** is largely single-threaded, the game engines—like SCUMM—utilize cooperative threading for limited concurrency, allowing for smooth gameplay execution without true parallelism.

Division of Responsibilities

Since ScummVM is an open-source project, there are no specific components developers are required to contribute to. While contributors are encouraged to work on open bug reports, anyone can clone the git repository and submit pull requests for changes to the software or optimizations for specific titles and platforms.

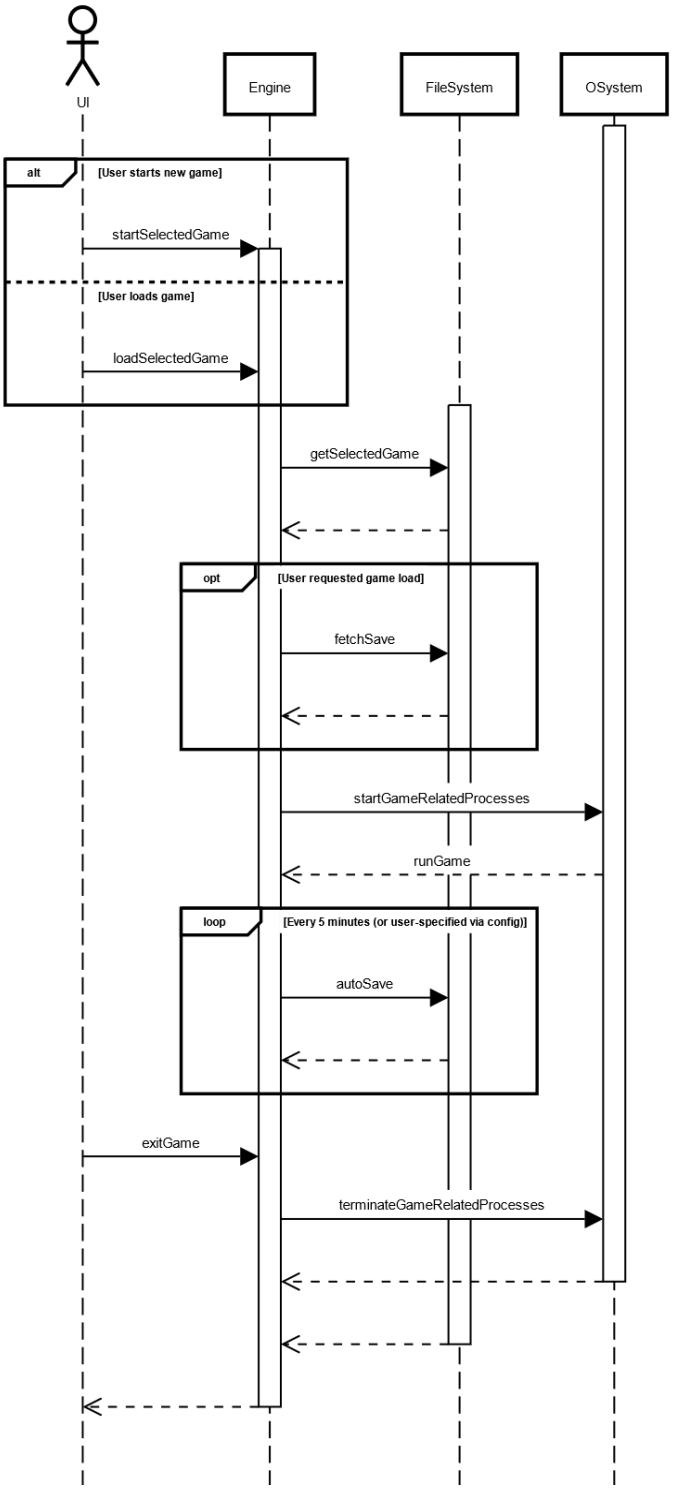
Despite this, developers still maintain a few responsibilities while writing new code. In order to ensure proper cohesion and style across the whole project, developers must ensure they write according to the standard coding style and commit messages such that they are all homogeneous.

These requirements include hugging braces, whitespaces in operators and “common sense” for coding style and through, specific and detailed messages for commit requests. Other coding conventions are also enforced in order to ensure ScummVM remains a portable program compatible with many different platforms (including little endian and big endian hosts).

To account for file management across multiple devices, contributors are forbidden to include file reading commands standard with C++. Instead, developers are required to use APIs provided by ScummVM for file management, although this issue mostly pertains to frontend, rather than backend.

Sequence Diagrams

Running a game in ScummVM



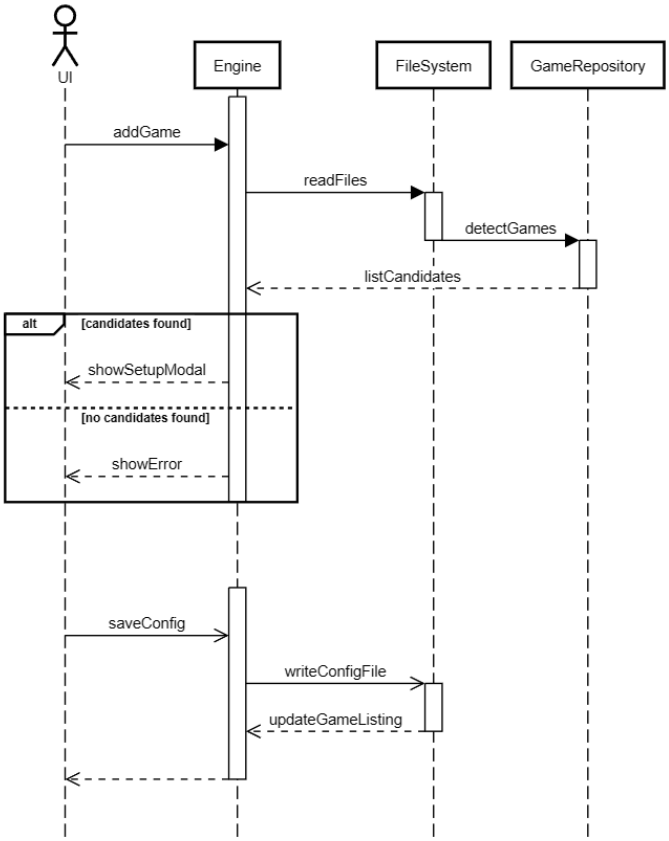
The diagram to the left demonstrates a very rough outline of how a game is run on ScummVM. It also illustrates the following use case: “The user starts or loads a game from their library, plays for an indefinite amount of time, then chooses to exit the game from the menu.”

To start a game, the user simply selects a game from their library, then clicks “Start Game” or “Load Game” from the side panel of the main launcher menu. The application engine will then find the selected game on the system and (optionally fetching the save data from the filesystem and into memory), will ask the OSystem/game engine to start the necessary processes for running the game. Afterward, the engine receives these processes and displays the game, audio, etc. to the user, and begins accepting input.

Every 5 minutes (or any time specified by the user), the game will auto-save. While not in the scope of this diagram, the user may also choose to save via the Global Main Menu.

Once the user is finished, they may either exit back to the launcher or completely exit ScummVM via the Global Main Menu, which will signal to terminate any necessary processes and the clearing of any necessary memory.

Adding a game to ScummVM



This diagram demonstrates the general flow of choosing a game from a user's hard drive, and "importing" it into ScummVM. It is loosely based on the actual source code.

When adding a new game via ScummVM's interface, you are prompted to choose a directory from your computer's file system. Each file in the directory is then searched by ScummVM, which stores a massive internal repository of games and their expected data files. If a game was successfully found, the application returns a list of potential variants or "candidates", which the user can then select from.

Following this, the user can configure a variety of emulation settings (graphics, audio, key mappings, etc) that will only be applied to the uploaded game. After pressing "OK", these settings (including the file path to the selected game) will be permanently saved to ScummVM's app data folder, and the game will be listed on the in-application library.

System evolution

ScummVM's evolution is driven by developments in technology, the expansion of goals beyond the original scope of the project and the community that continues to use and update the software.

History of ScummVM's evolution:

All information in this section can be found in the release notes: https://docs.scummvm.org/en/latest/help/release.html

In 2001, ScummVM was launched specifically to emulate the SCUMM (Script Creation Utility for Maniac Mansion) engine that was used by LucasArt who released 15 adventure games from Labyrinth in 1986, to Escape from Monkey Island in 2000.

From 2002 to 2006 is when ScummVM experienced its most significant developments. ScummVM extended its support to many more platforms like Windows, Linux, macOS, iOS, Android, and even niche devices like PlayStation Portable (PSP), Nintendo DS, and Raspberry Pi. During this time they also started to add various features to enhance the gameplay:

- Support for high resolution graphics.
- Better GUI, making it easier to configure and manage multiple games.
- Voice and Soundtrack Support.
- The ability to change original game scripts which allows for bug fixing and community modding.

Most notably however, during this time period, ScummVM began expanding beyond its original scope to add support for new game engines like the Sierra's AGI and SCI engines, Coktel Vision's Gob engine, and various others.

From 2007 and onward, the evolution of ScummVM was almost entirely focused on continuously adding support for more games, bug fixing and improving on current features.

Perhaps the largest steps that happened during this time was in the past few years when ScummVM integrated ResidualVM, a sister project focused on running 3D adventure games. This allowed ScummVM to now expand its support to 3D games too, opening up a whole other genre of adventure games.

How ScummVM Can Continue to Evolve and Conclusion:

ScummVM has far outgrown its original goal of being a tool for playing SCUMM games. It is now a general purpose interpreter and game preservation platform. Its continued evolution can be driven by several factors:

Supporting more game engines: ScummVM can evolve by incorporating additional engines like older FPS (First Person Shooter) or RPG (Role Playing Games) engines, text-based interactive fiction engines, or hybrid genres.

Better 3D game support: 3D game support for ScummVM is still relatively new and there is plenty of room for improvement on that front. This could include better 3D rendering and support for more 3D adventure game engines.

Improving game preservation efforts: Most of the games that ScummVM offers support for are truly pieces of video game history. ScummVM can choose to embrace this and focus its goals on archiving the games' assets, scripts, and source code. Additionally, ScummVM could choose to partner with online video game archival companies like the Video Game History Foundation.

References

"Developer Central." *ScummVM :: Wiki*, 10 August 2023, https://wiki.scummvm.org/index.php?title=Developer_Central#For_Backend_authors

"Release notes — ScummVM Documentation documentation." *ScummVM Documentation*, https://docs.scummvm.org/en/latest/help/release.html

"ScummVM", https://www.scummvm.org/

"ScummVM GUI Framework Overview." *The ScummVM Blogs*, https://blogs.scummvm.org/av-dx/2021/06/13/scummvm-gui-framework-overview/

"ScummVM main repository." *GitHub*, https://github.com/scummvm/scummvm