

ScummVM Architecture Report

Authors:

Andrew Clausen	21awc7@queensu.ca
Arda Utku	21au10@queensu.ca
Arianne Nantel	20aan8@queensu.ca
Daniel Dousek	22dd1@queensu.ca
Kyra Salmon	21kms22@queensu.ca
Oliver Macnaughton	20owdm@queensu.ca

Abstract:

In this report, we discuss the overarching architecture of ScummVm and the SCI engine. To begin, the ScummVm team is structured into six different parts. Namely the volunteer developers (devs), the engine teams, backend and platform support, subsystems and infrastructure, website and documentation, and lastly, the game and GUI translation team. Secondly, ScummVm depends on volunteer contributions for the code that allows the different games to be played. To ensure a smooth process there are protocols put in place to push code changes onto the platform. These protocols ensure changes made don't disrupt the preexisting code. Thirdly, the scummVM and the SCI engine can be represented by specific architectural styles, the layered style and the interpreter style respectively. The layered style best describes the ScummVm architecture as ScummVm can be divided into an organized hierarchy where each layer has its own responsibilities and functionality. These layers are: the Presentation Layer, the Operating System, the Abstraction Layer, the Logic Layer and the Data Layer. ScummVm also benefits from the abstractions between the different functional parts that the Layered architectural style provides. The interpreter style best describes the SCI engine as it allows source code for older games to be run on a new platform, one that it wasn't initially designed for. Finally, scummVM is looked at through the view on concurrency. This leads to an interesting discovery, concurrency is not used often in ScummVm's tasks. However, a form of multithreading, cooperative threading, is used in the SCI engine for certain tasks such as audio processing. Diagrams and figures are used throughout the report to better show how different components work together in both the ScummVM architecture and the SCI engine. Following the architecture, the report dives deeper into how the external interface interacts both with the user and the different layers of the architecture.

Introduction and Overview:

ScummVM is a volunteer developed program which allows users to run over 325 classical graphical adventure games on systems they weren't designed for. It is free provided that the user already possesses the data files for the game. In this report, we attempt to complete a deeper dive into the backend of this virtual machine.

Firstly, an overview of the 6 divisions of labour between the volunteer developer teams and their roles; These are project leadership and administration, engine teams, backend and platform support, subsystems and infrastructure, website and documentation, and lastly, the game and GUI translation team. Each team is vital to the development and expansion of ScummVM.

The architecture of ScummVM is best described as layered, consisting of 5 different layers. The presentation layer, acting through the GUI, is responsible for the parts of the system that the user sees visibly on the screen and is able to interact with directly. The operating system layer, as the name suggests, manages resources within the operating system, including those needed by the presentation layer. The following layer is the abstraction layer. This permits communications between the operating system from the above layer and the SCI engine. The logic layer communicates with the SCI engine to form the game engine. It completes all tasks related to the behavior of the game. The fifth and final layer is the data layer. This layer holds all the essential files and components of the user's selected game.

The layered architecture seen in ScummVM is also ideal for its necessary non-functional requirements. Due to the very nature of ScummVM, portability is an indispensable necessity. This feature is made possible via the OSSystemAPI, which abstracts away the details of different operating systems to ensure compatibility. Additionally, a second critical NFR for ScummVM, is performance. Users will have no interest in the system if games experience excessive latency, thus a special focus is reserved for the FPS management and communications between components.

The aforementioned SCI engine is also described in detail in this report. The Sierra Creative Interpreter or Script Code Interpreter has a number of subcomponents that are essential to the cohesive function of the virtual machine. It usefully handles tasks such as internal state interpreter, program state, logic resources and graphics and audio. Despite being within the layered style of the primary architecture, the SCI engine functions with an interpreter style architecture.

An interesting discovery within this research relates to the concurrencies within the implementation of ScummVM. While some concurrency and instances of multithreading are present, they are fairly rare. Since the majority of games supported by ScummVM are older, they were created before multithreading was industry standard and therefore it would be counterproductive for the virtual machine that is running them to be anything other than single-threaded. That being said, ScummVM does have a secondary, low level thread which can concurrently execute tasks such as audio processing.

Overall, these many components to ScummVMs architecture and developer efforts are what creates such a fantastic, free, user experience for those who want to play nostalgic games on their current machines.

Architecture

Responsibility among participating developers

In the ScummVM project, the success of its development relies on the effective deviation of responsibilities amongst six key divisions. From high-level leadership and engine teams to platform support and documentation, each division has an important

part in the overall structure of the project. The project leadership division is responsible for overseeing strategic direction, while the engine division focuses more on maintaining and improving the individual game engines. The backend division focuses on platform compatibility, and the subsystem division ensures the infrastructure's reliability. Meanwhile, documentation and website divisions handle the public-facing aspects, and the translation divisions allow accessibility across multiple different languages.

The highest-level and most important divisions in the ScummVM project are the project leadership and administration divisions. The project leadership division is responsible for overseeing the overall direction, coordination, and decision-making for the project. Some of the high-level responsibilities include managing the project's roadmap, ensuring quality control, and coordinating the efforts of various teams. The administration division handles public relations, communication, and organizing all resources needed for the project.

The second highest division is the engine teams. ScummVM's development is structured around dedicated teams, each responsible for maintaining and enhancing specific game engines like SCUMM, AGS, Blade Runner, and many more. These teams ensure their assigned engine operates smoothly within the ScummVM framework, addressing bugs and performance issues as needed. This modular setup allows experts to concentrate on the distinct needs of each game engine, ensuring specialized care and attention.

The third highest division is the backend and platform support team. The backend teams in ScummVM are responsible for ensuring that the project runs seamlessly on a wide range of platforms such as IOS, Android, and Nintendo. Their primary task is to adapt the core ScummVM engine so that it is compatible with various operating systems and hardware environments. This process involves addressing the unique challenges that each platform presents such as processing power and input methods. By improving the engine to account for these variables the backend team plays a crucial role in expanding the usability of ScummVM. They are also responsible for ensuring that updates to the ScummVM engine are consistently applied across all supported platforms.

The fourth division is the subsystems and infrastructure team. Developers in this division manage the various subsystems in ScummVM, such as file APIs, sound mixers, and the graphical user interface (GUI), which form the internal infrastructure that supports the game engine. These subsystems are crucial for ensuring that ScummVM runs effectively across platforms. Developers in this division focus on optimizing performance, adding new features, and maintaining the essential tools that all the engines and platforms rely on. Their work ensures that the foundational components of ScummVM operate smoothly, enabling the game engines to function properly regardless of the platform.

The fifth division is the website and documentation team. This team handles the website, documentation, and all public-facing resources which all play a vital role in keeping up to date with the community and communicating between developers and users. This team maintains the website making sure it is updated with all the latest information on releases, updates, and development progress. They are also responsible for keeping documentation of everything going on, which is an invaluable resource for developers contributing to the project and users trying to navigate it. Public-facing

platforms, such as forums, are also managed by this division. This provides a space where users can ask questions and report any issues they come across. By ensuring that these resources are clear, accurate, and up to date, this team plays a crucial role in keeping the community engaged and informed, while also making it easier for new developers and users to get involved with ScummVM.

The lowest-level division in the ScummVM project is the Game and GUI translation team. These teams are responsible for translating game content and the user interface into different languages to accommodate anyone who wants to play.

All of these divisions are each responsible for their own work and sections of the system but all play an important role working together to achieve an efficient and smooth system.

How the code is done between the developers

Scumm development is done by dividing ownership and responsibilities among its developers. Each developer is given full ownership of their area of expertise such as a specific engine or package, where they can freely push code while following the developer guidelines. This range of freedom allows developers to make decisions and changes without constant oversight or approval for minor modifications. For more significant changes or updates where these usually affect areas outside of their ownership communication and collaboration remain essential. Developers must discuss, collaborate and seek the approval of other developers to ensure that the entire codebase remains stable.

The developers' guidelines allow the codebase to be readable and maintain structure. Coding standards are followed to maintain consistency in the project such as header comments on large files or file-level comments on large or confusing functions. Any commits to the repository must begin in all capitals stating the location of the change followed by a descriptive action with the change.

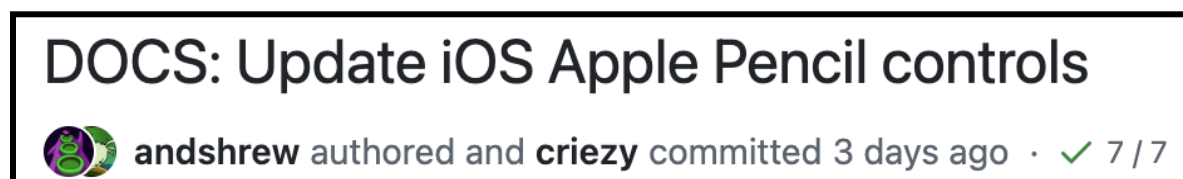


Figure 1. This image shows an example of a GitHub commit for a new change

Here you can see the location being Docs and the action is an Update with the change to the Apple pencil controls. Using this guideline also minimizes the need for constant code refactoring, as well-structured, documented, and standardized code reduces errors in future iterations.

The Scumm team communicates using Discord, Github, and a mailing list. Discord is primarily used for team updates, development discussions and advice through group messages and calls. Github from a communication level is primarily used for reporting bugs found in Scumm and tasks that need to be completed. The mailing list is used to make announcements that need to reach all developers such as long-term updates or policies.

Before a developer joins the team they are required to demonstrate their skills by submitting PRs on relevant ongoing issues. These issues are minor fixes but it allows

the development team to assess the skills of the developer. The PR is kept open for 2 weeks to be reviewed by other developers to ensure quality of code, adherence to coding standards, and general compatibility with the project's long-term goals. Reviews provide feedback and address any concerns. After this process is repeated a few times and the developer has shown their skills they are given full access.






To ensure all significant code changes are compatible and will not break the build. The Scumm developers use a Buildbot System. The Buildbot system is a vital tool that helps developers maintain the integrity of the code base. With any significant commit to the codebase, the Buildbot automatically compiles and tests the updated code to identify any potential issues or breakages. The Buildbot system allows for continuous integrations where automation and testing happen frequently adapting dynamically.











Build steps	Build Properties	Worker: builder-2	Responsible Users	Changes	Debug
<div> 🔍 🔍 All master-raspberrypi/12130 4:18 build successful SUCCESS Triggered from:fetch-master/10918 </div>					
0	worker_preparation	4 s	worker builder-2 ready		
1	locks_acquire	1 s	locks acquired		
2	clean	0 s	clean (skipped)		
3	check configure.stamp freshness	0 s	generated file is newer than source file		
# 4	configure	0 s	configure (skipped)		
5	compile	2:39	compile (warnings)		
6	test	1:33	test (warnings)		
7	package	0 s	package (skipped)		
8	upload package	0 s	uploaded (skipped)		
9	link latest daily build	0 s	linked (skipped)		

Figure 2: This image shows the Buildbot process where warnings are flagged and certain steps are skipped if specific conditions are met, such as when files haven't changed since the last build or when a previous successful build allows for caching of certain components.

The Scumm model also uses another CI/CD pipeline with github actions. The github actions are triggered on certain events such as a push to main or a commit to a branch. The pipeline compiles the code, runs tests, and ensures that no new changes introduce breaking errors. If an issue arises, the pipeline notifies the responsible developer, who must resolve the problem before the changes can be merged.

Lastly the Scumm team uses github branches which are named in a structured format to track different versions. For example, branches like branch-2-8-1-android or branch-2-7-0 follow a clear versioning system. The main branch, master, is the default where most stable changes are merged. Version-specific branches like 2-8 or 2-7 help developers track changes related to particular releases. This structure simplifies code management, making it easy to address issues or develop new features without disrupting the stable version. Developers work on their changes in these branches, ensuring smoother integration and conflict resolution when merged into master.

Default					
Branch	Updated	Check status	Behind	Ahead	Pull request
master  	 12 minutes ago	 2 / 7		Default	 ...

Active branches					
Branch	Updated	Check status	Behind	Ahead	Pull request
branch-2-8-1-android  	 last month	 6 / 7	8966	696	 ...
branch-2-8  	 2 months ago	 6 / 7	8966	695	 ...

.Figure 3: This image shows an example of what a the scummVm repository branches look like

Style

In application, ScummVM can be viewed as having a layered architecture, as seen in the table below. In this abstraction of system, each layer communicates directly with its adjacent layers

Presentation Layer <p style="text-align: center;">GUI</p> <p>Seen as the ScummVM launcher or running game. This layer is responsible for the user interface, and allows the user to interact with the application. The presentation layer is coordinated by the operating system, with possible help from rendering or graphics libraries such as OpenGL or SDL.</p>
Operating System <p style="text-align: center;">MacOS, Windows, etc.</p> <p>Manages operating system resources such as window management and input handling (resources needed by presentation layer.) Implements platform-specific tasks sent by the Abstraction layer, such as rendering updates to state in GUI. Outputs commands and resources for rendering and presenting information.</p>
Abstraction Layer <p style="text-align: center;">OSystem API</p> <p>Allows communication between SCI engine and operating systems, handles platform specific commands, essentially converting output of logic layer into instructions that can be interpreted by the operating system. OSystem API is dependent upon ScummVM “backends”, the code that instantiates the API for a specific OS, allowing the game engine to receive input from and send commands back up to the operating system.</p>
Logic Layer <p style="text-align: center;">SCI Engine</p> <p>Game engine, responsible for interpretation of game files, processing of game logic, tracking of game state, handling events (button press, mouse click, etc.). Essentially controls the behavior of the game and how it runs. Sends output (new game state) to OSystem API to be represented in GUI. Receives input from user-loaded game files and OSystem API calls.</p>

Data Layer

Game Files from Local System

Game files, scripts, animations, sprites, graphics, all specific to the game. The game files are user provided, and ScummVM must be given access to the location of the users local game files in order to load them into the engine. The data layer represents the building blocks of the game that are used in order to run it.

As a program which allows users to run games from a wide range of developers on many different platforms, one main goal of the ScummVM architecture is to provide high levels of portability. The ability for ScummVM to be ported to different platforms is dependent upon a series of platform-specific backends that register an API (OSystem) that allows the device on which ScummVM is being run to communicate with the game engine. The OSystem API is crucial for portability as it abstracts away the physical operating system, allowing the game engine to determine a new game state, interpreting it, and then interacting with the rendering components of the OS, as well as animations, textures, etc. stored in the game files to display the changes on the monitor. The main components at play here are the user GUI, through which the user interacts with the system, user-side game files, OSystem API which is facilitated by the platform-specific backend component, the operating system, and the game engine component (SCI engine).

In general, the level of portability of the system can be determined by the number of platforms it can be run on, or the platforms that have a platform-specific backend code in the ScummVM database. Thus, it is up to the community of developers to generate more of these backend codes to expand the portability of ScummVM.

Another important requirement for the system is performance. A drop in performance in the context of ScummVM could be defined as a drop in FPS (frames per second) which could be caused by a variety of issues including over-rendering (surplus of draw calls), or an overloading of local CPU. In this context, the implementation of the original game engines (rewritten in C++ for compatibility reasons) is crucial in determining how much the host OS is interacted with. Developers must ensure that the rate at which the engine gives instructions to the OS is manageable, and can be rendered at a rate that is equivalent to the implementation of the original game.

Aside: Subcomponents of an SCI Engine

In the layered structure described above, a component that is worth explaining in more detail is that of the game engine. In this project, we look specifically at an implementation of ScummVM that uses the SCI engine (Sierra Creative Interpreter, or Script Code Interpreter). Since the engine is crucial in the operation and performance of the application, a look into its subcomponents and conceptual architecture is warranted in order to better capture how ScummVM operates. The conceptual architecture of the SCI engine can be well described as that of an interpreter. In doing so, we bring to light a few submodules such as “Program State”, a subcomponent of the game engine which keeps track of the current game state, and an “Internal State Interpreter”, which processes and manages the game’s state and logic (provided by game scripts, the main component being interpreted). Upon initialization of the engine, a timer is set up which

“ticks” 60 times per second. This timer allows for execution of components such as an input manager or sound play (abstracted away in the diagram below), and also informs the game clock.

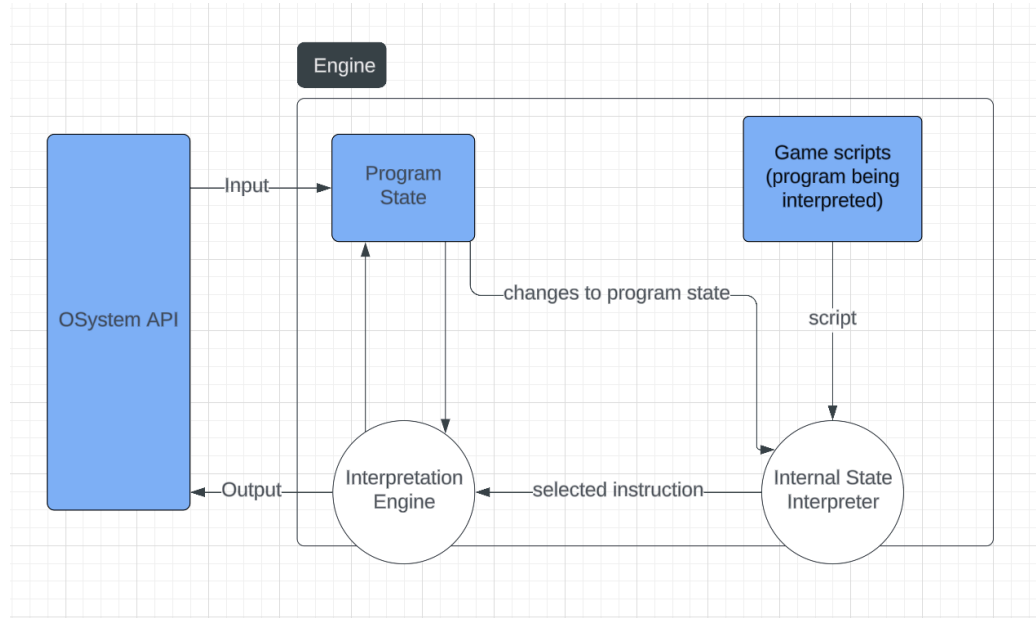


Figure 4: A box and arrow diagram showing how the SCI engine in the Interpreter architecture style and how it interacts with the OSystem API

The above figure provides an abstracted view of the SCI engine used to run the game in this context. In reality, the subsystems involved in the SCI engine are those involving graphics rendering, audio output, computation (PMachine), window manager, event manager, parser, and music player. Resources in SCI are divided into four categories: Graphics, Audio, Logic, Text. During the startup process the memory heap is initialized, the configuration files parsed, the various subsystems and managers initialized, the machine state is saved for restarting the game, and the game object is collected and run by execution of the play method. The SCI engine has an object-oriented setup that allows developers to modularize data and behaviors, allowing for code and component reuse within and across games. To execute code, script resources must be loaded onto a heap and then operated on. In this context, the heap is the only space in memory that the virtual machine can operate on directly.

Another feature important to ScummVM is the ability to save games progress locally, such that the game can be reloaded and played from where the user last left it. This process involves saving the game state on the host machine and loading it into the engine upon resumption of the game. The process of saving and loading game states is illustrated in the sequence diagram below.

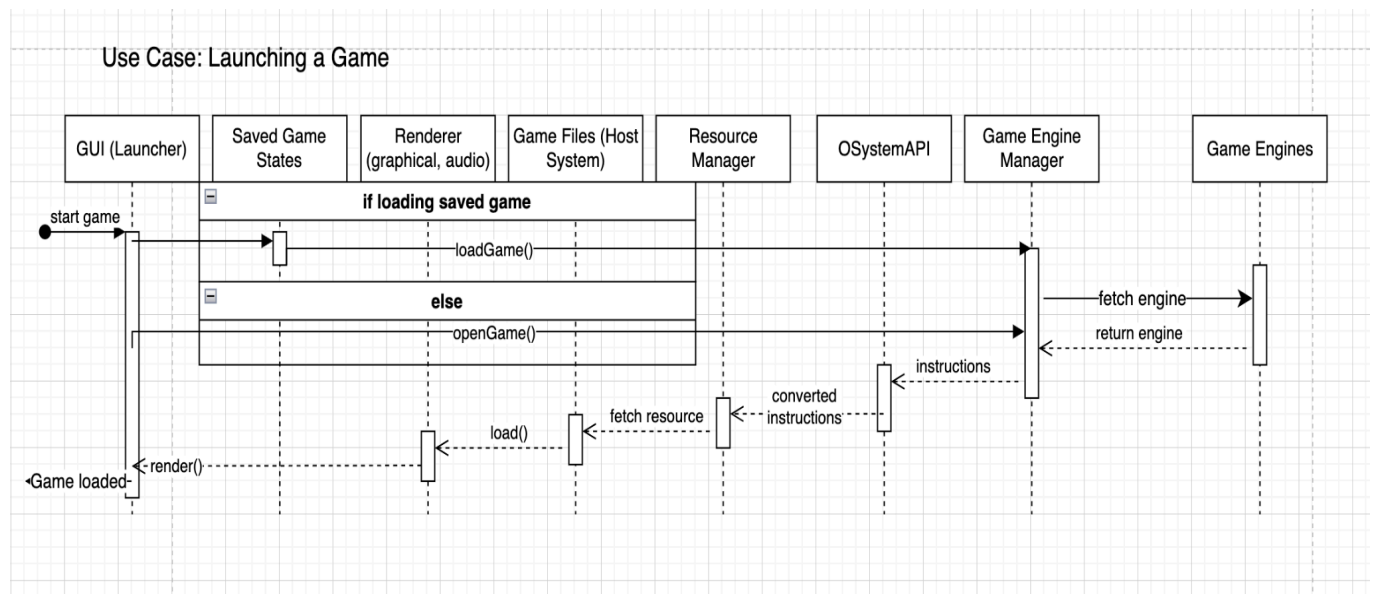


Figure 5: This UML sequence diagram shows the use case of loading a game from a saved state.

Concurrency:

Concurrency, which covers the methodologies used to have a program perform multiple tasks simultaneously, exists in various auxiliary forms in ScummVM. Since ScummVM is a virtual machine designed to run a diverse variety of older video games, it would be unproductive to try and implement a form of core level concurrency. This is a compatibility concern as many of the titles which can run on ScummVM were created prior to multithreading becoming a standard in the industry. As such, ScummVM is primarily a single-threaded application. However, ScummVM does utilize cooperative threading for secondary tasks such as audio processing, networking, and resources loading and decompression.

Cooperative threading is a form of concurrency where multiple threads execute simultaneously within a shared environment by explicitly yielding control to one another. The yielding of control occurs in accordance with four states that each thread can be in, which include RUNNING, PENDED, DELAYED, and FROZEN. Each thread goes into a stack where the thread in execution is at the top of the stack and RUNNING or DELAYED, where it is waiting for its allot time slot to expire, and any parent threads are PENDED, next in the stack. Threads can also be FROZEN while waiting to regain control. In ScummVM, this is implemented by allowing instructions to execute out of a single script until it has to suspend, in which case the flow of execution will move onto the next script. Foregoing some exceptions, each script has a chance to run once per frame. This is a simpler implantation of concurrency since it reduces the complexity of synchronization between scripts, and is easier to implement within the virtual machine without having to rely on operating system level threading. ScummVM is an interpreter which means it runs games primarily through the sequential executions of scripts written in the Scumm Scripting language. This form of multithreading is able to aid in that execution and enhance the performance of the engine without adding severe complexities which could negatively affect compatibility and stability.

As mentioned previously, cooperative threading allows certain tasks, such as audio processing, to be executed on a separate thread to the main execution of the program. ScummVM utilizes a dedicated thread to handle low-level audio processing and playback. ScummVM uses an API called SDL (Simple DirectMedia Layer) which deals with mixing audio channels and buffering audio data. This audio data is buffered on its own dedicated thread in the background, which allows the audio to run concurrently while the game engine's main logic executes on the main thread. The processed audio data is then sent to the operating system's own audio output API, which then deals with outputting the audio for the user to hear. In addition to the low-level audio processing handled by SDL, ScummVM uses the MIDI protocol to deal with high-level audio management such as device output control.

External Interfaces:

Client side interfaces include the Graphical User Interface, through which the menu and game state are displayed. Graphics instructions are sent to the user-side renderer by the SCI engine based on the most recently made updates to the game state. OSSystem (the interface between the OS hardware and game engine) is responsible for rendering the sent image data to the correct positions on screen. Along with position updates and frame data, the SCI engine also sends timing information such as frame delay to control animation speeds. Here, another important aspect of performance is brought to light. Management of FPS can have direct effects on quality of gameplay and smoothness of graphical transitions. In addition, since rendering each frame takes some computation power, enforcing frame delay may be essential in order to not overload the system's processing units. Thus, developers must find an appropriate middle ground between having too high of a frame delay (leading to stuttering gameplay) or too low (leading to an overload of the systems processing units).

Another external interface would be the game codes that the user must give ScummVM access to in order for it to run. The game files act as an input for the SCI engine, providing data such as sprites, backgrounds, audio, scripts, and possibly save files. Since the files are manually loaded into ScummVM upon launch of a game, they can be recognized as an external interface to the system.

Finally, ScummVM allows for game files to be shared either across a cloud service or local web server, such that more than one person/device can access game files and config files. For cloud devices, ScummVM supports DropBox, OneDrive, GoogleDrive, and Box. Saved games files automatically synchronize when the game is launched, saved, and reloaded from save. This functionality is built into the ScummVM interface, as a local web server can be initialized through the ScummVM launcher. A url is provided, and once a user types that url into a web browser, they are taken to an index of all files stored on the server which can be downloaded from. Configuration options for the server can be set in Launcher (which files are available, which directory, etc.).

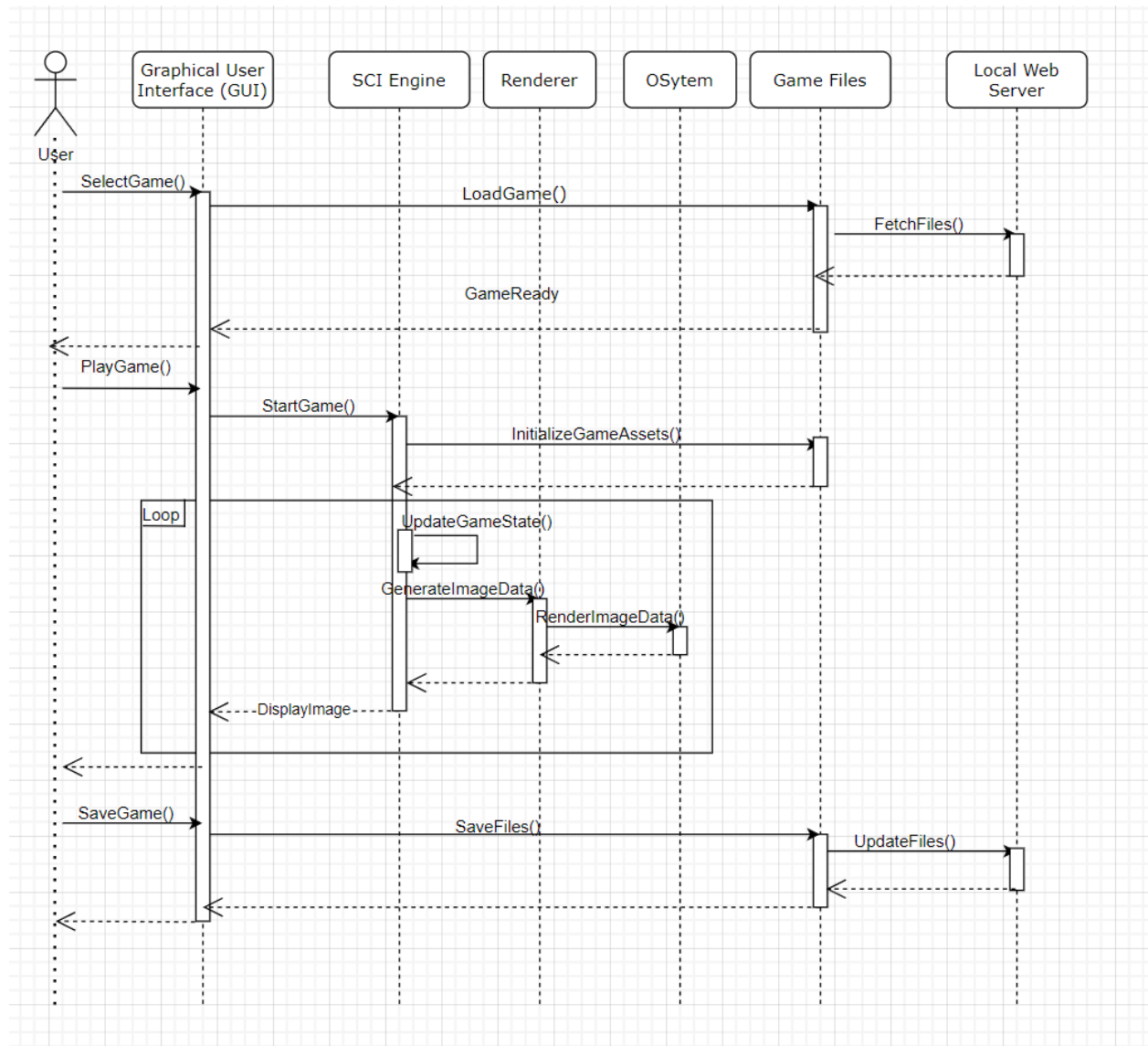


Figure 6: This UML sequence diagram shows the rendering of a game during play

Use Cases:

Use Case 1: User loads and starts a game using ScummVM

Sequence of Steps:

1. [Client Action] The user launches Scummvm and selects a previously purchased game from their files via the GUI.
2. [Presentation Layer] The operating system is used to call and render the ScummVM launcher's GUI. It will display a game list and accept the user input.
3. [Abstraction Layer - OSytem API] Once a game is selected, OSytemAPI is responsible for generating platform-specific commands to aid the game engine in execution.

4. [Logic Layer - SCI Engine] The game logic and state is initialized by the SCI engine which then begins to interpret game scripts and behaviors (such as actual gameplay or simply menus).
5. [Data Layer] Game files, (scripts, graphics, and audio,) are loaded into the data layer from the users machine. These resources are then sent up a layer (to the SCI engine) to be processed so the game can be run.
6. [Presentation Layer] The entire game has now been rendered on screen and is ready to be played by the user!

This is a use case for the layered architecture, demonstrating how the various layers work together and interact. Thus allowing the game to run smoothly and the user to be satisfied.

Use case 2: A User saves their game progress

Sequence of Steps:

1. [Client Action] The user opens the in game menu, pausing the game
2. [Presentation Layer] The GUI shows the user the game menu where a save button can be found.
3. [Client Action] The user presses the save button sending a message to the operating system
4. [Operating system] Informs the abstraction layer that the user wants to save their progress
5. [Abstraction Layer] Informs the logic layer that the user wants to save their progress
6. [Logic Layer] The game progress is recorded. Meaning the sci engine records the game states and any other information required by the game (inventory, player coordinates, ect..)
7. [Data Layer] The game files are saved on the users computer in the location given to ScummVM.
8. A message is sent back up the chain through every layer to update the GUI to show that the game has been saved

This use case demonstrates how the simple action of saving a game will be executed through the different layers present in the ScummVm architecture.

Data Dictionary:

- Layered Architecture: The layered architecture style is best for applications that can be divided into an organized hierarchy of responsibilities or functions. Each layer has one specific job and they only allow neighboring layers to communicate allowing for abstractions when necessary.
- Interpreter Architecture: The Interpreter architecture style is best for applications where the best language or machine to use is not available for the current system.
- OSsystem API: Defines what a game can use (whether a user can draw on a screen, use mouse clicks, ect..). It also shields the SCI engine from the OS system itself. This means

- Concurrency: When a system runs multiple processes at the same time
- Multi-threading: When many threads (parts of processes) are executed in fast succession so that they appear to be executed in parallel, or concurrently
- Single-threaded: When a system executes one thread at a time in sequential order with no concurrency
- SCI Engine: Game engine, responsible for interpretation of game files, processing of game logic, tracking of game state, handling events (button press, mouse click, etc.).
- Portability: Probability is an NFR that refers to how easily an application can be executed on a platform it was not designed for
- Performance: Performance is an NFR that encompasses, response times, throughput, and deadlines, to define how quickly an application runs
- Cooperative threading: Is a form of concurrency where multiple threads execute simultaneously within a shared environment by explicitly yielding control to one another.

Naming Conventions:

- NFR: Non-functional Requirements
- ScummVM: Script Creation Utility for Maniac Mansion Virtual Machine
- GUI: Graphical User Interface
- SCI: “Sierra Creative Interpreter” or “Script Code Interpreter”
- FPS: Frames per Second
- OpenGL: Open Graphics Library
- SDL: Simple DirectMedia Layer
- Devs: Developers
- OS: Operating System
- API: Application Programming Interface
- MIDI: Musical Instrument Digital Interface
- AGS: Adventure Game Studio

Conclusions:

In conclusion, the scummVM team is composed of 6 different teams all working towards the goal of having an open source and free game platform that allows users to run over 325 classical graphical adventure games on systems they weren’t designed for. Volunteers can learn about the platform and submit code modifications or improvements which allows ScummVm to continuously grow and improve as technology evolves. The scummVm platform is best described by a layered architecture where each layer has its own functionality. The SCI engine is best described by the interpreter architectural style allowing for the games to be played on a variety of different platforms no matter which platform they were initially designed for. In future assignments, we will explore the concrete architecture of ScummVm and the SCI engine to gain a better understanding of how every part and layer works together to create a seamless gaming experience.

Lessons Learned:

Through the process of researching and learning about the ScummVM we learned the importance of coordinating research and the discussion of our findings with the group. When multiple people are investigating a large system together, it seems best to strike a balance between separating research topics and giving everyone a comprehensive understanding of the system. Dividing the research between group members helped the efficiency of the information collection, as one member could communicate to the group a synthesized version of what they learned. It is beneficial for all parties to ask questions and provide feedback to the presenting members. During the presentation of information, we learned it is best to convey the facts before providing your interpretation of them. When facts are conveyed first, it allows all members to begin to form their own interpretation based on their own research or class material, helping facilitate discussion and creation of alternative solutions, or consensus on the presented interpretation. In general, coordination and communication between team members (dividing up material, filling knowledge gaps, etc.) is crucial to the success of such a project.

References:

- "Class 17: Concurrency." MIT 6.005, Massachusetts Institute of Technology, <https://web.mit.edu/6.005/www/fa14/classes/17-concurrency/>. Accessed 7 Oct. 2024.
- "Concurrency: Synchronization." Queen's University, <https://onq.queensu.ca/d2l/le/content/959322/viewContent/5711375/View>. Accessed 7 Oct. 2024.
- Folmer, Eelke. Component-Based Game Design. Eelke.com, <https://www.eelke.com/assets/pubs/cbgd.pdf>. Accessed 7 Oct. 2024.
- "Game Engine Development for the Hobby Developer, Part 2: Engine Parts." Indie Game Dev, 15 Jan. 2020, <https://indiegamedev.net/2020/01/15/game-engine-development-for-the-hobby-developer-part-2-engine-parts/>. Accessed 7 Oct. 2024.
- Groebelsloot, Jasper. "Design of a Point-and-Click Adventure Game Engine." Groebelsloot.com, 1 Dec. 2015, <https://www.groebelsloot.com/2015/12/01/design-of-a-point-and-click-adventure-game-engine/>. Accessed 7 Oct. 2024.
- "Handling Different File Types in ScummVM?" ScummVM Forums, 25 Aug. 2009, <https://forums.scummvm.org/viewtopic.php?t=7886>. Accessed 7 Oct. 2024.
- Metta, Naveen. "Single-Threaded vs. Multi-Threaded Programs in Java: A Comprehensive Comparison." Medium, 11 May 2023, <https://naveen-metta.medium.com/single-threaded-vs-multi-threaded-programs-in-java-a-comprehensive-comparison-8c294dcc6c9d>. Accessed 7 Oct. 2024.
- "MIDI Overview." Center for Electronic and Computer Music, Indiana University, <https://cecm.indiana.edu/361/midi.html>. Accessed 7 Oct. 2024.
- Mozilla Developers. "OpenGL Glossary." Mozilla Developer Network (MDN), <https://developer.mozilla.org/en-US/docs/Glossary/OpenGL>. Accessed 7 Oct. 2024.

"Multithreading in Java." Queen's University,
<https://onq.queensu.ca/d2l/le/content/959322/viewContent/5711378/View>.
Accessed 7 Oct. 2024.

ScummVM buildbot <https://buildbot.scummvm.org/> Accessed 7 Oct. 2024.

ScummVM Contributors List. GitHub,
<https://github.com/scummvm/scummvm/blob/master/AUTHORS>. Accessed 7
Oct. 2024.

ScummVM Documentation Team. ScummVM Documentation. ScummVM,
<https://docs.scummvm.org>. Accessed 7 Oct. 2024.

ScummVM Wiki Contributors. "Developer Central." ScummVM Wiki,
https://wiki.scummvm.org/index.php?title=Developer_Central. Accessed 7 Oct.
2024.