CISC 322
Assignment 1
Conceptual Architecture of ScummVM
Oct 11, 2024

Group-10

Antony Li

Ghaith Khalil

Owen Gimbel

Yuchen Lin

Mitchell Nagler

Jasmine Zangeneh

Table of Contents

Abstract

This technical report presents an analysis of the conceptual software architecture of ScummVM, an open-source project that enables users to play classic point-and-click adventure games on modern systems [1][2]. Our conceptual architecture was derived from the project's source code and documentation [3][5]. We identified that ScummVM utilizes several architectural styles, including interpreter, plug-in, and layered styles. These architectures support the diverse range of game engines and platforms that ScummVM accommodates. We also discuss the non-functional requirements, external interfaces, and use cases that are critical to ScummVM's functionality.

Introduction and Overview

As technology advances, many classic games become incompatible with modern hardware and operating systems. ScummVM addresses this issue by providing an open-source platform that reimplements the original game engines, allowing users to play classic point-and-click adventure games on contemporary systems [1][2]. Originally designed to run games using the SCUMM (Script Creation Utility for Maniac Mansion) engine developed by LucasArts, ScummVM has expanded to support numerous other game engines and titles [2][3].

ScummVM's primary goal is to preserve and enable the enjoyment of classic games, ensuring they remain accessible regardless of changes in hardware or operating systems [1][3]. By reimplementing the game engines, ScummVM allows the original game data files to be used without the need for the original executable code [4][5].

At its core, ScummVM provides a modular and extensible architecture that supports multiple game engines and platforms [5][6]. The system integrates key components such as the Engine Manager, Graphics Subsystem, Input Subsystem, Audio Subsystem, and File I/O [3][6]. These components work together to interpret game scripts, render graphics and audio, handle user input, and manage game data files [3][6].

Derivation Process

To derive the conceptual architecture of ScummVM, we examined the project's source code, documentation, and developer resources [3][5][6]. The ScummVM project is hosted on GitHub, which gives our group access to the full source code and commit history [5]. Additionally, we reviewed the ScummVM Wiki, developer forums, and official documentation to understand the system's structure and design principles [3][6][7].

Our analysis focused on identifying the main components, their responsibilities, and the interactions between them. We paid particular attention to how ScummVM supports multiple game engines, the mechanisms for platform abstraction, and the strategies used for code modularity and extensibility.

Through this process, we identified key architectural styles used in ScummVM, including the interpreter pattern for game script execution, the plug-in architecture for engine modularity, and layered architecture for platform abstraction. These styles are instrumental in achieving ScummVM's goals of portability, maintainability, and scalability.

Non-Functional Requirements (NFRs)

ScummVM's non-functional requirements are critical to its success as a multi-platform, multi-engine game interpreter. These requirements include:

Portability

- ScummVM must run on a wide range of hardware and operating systems, including Windows, macOS, Linux, Android, iOS, and others [1][6][7].
- The codebase should be easily portable to new platforms with minimal changes [5][7].

Extensibility

- The architecture must allow for easy addition of new game engines to support more games [5][6].
- Developers should be able to add support for new platforms without significant rework [5][6].

Maintainability

- The code should be organized in a way that is easy to understand, maintain, and update [5][6].

- Modular design should allow developers to work on individual components without affecting others [5][6].

Performance

- ScummVM should execute games efficiently, providing smooth gameplay experiences even on less powerful hardware [1][3][6].
- Resource usage (CPU, memory) should be optimized [3][6].

Compatibility

- ScummVM must accurately emulate the behavior of the original game engines to ensure that games run as intended [3][4][6].
- High fidelity in graphics, audio, and gameplay mechanics is essential [3][6][14][15].

Usability

- The user interface should be intuitive and easy to use for both novice and experienced users [6][12][13].
- Configuration and setup should be straightforward [6][12][13].

Reliability

- ScummVM should run games without crashes or errors [6].
- It should handle unexpected situations gracefully, providing meaningful error messages [6][12].

Modifiability

- ScummVM should be able to be changed to support new platforms and games over time without affecting performance or portability [5][6].
- Expanding the scope of the project should not interfere with other games and engines [5][6].

Conceptual Architecture

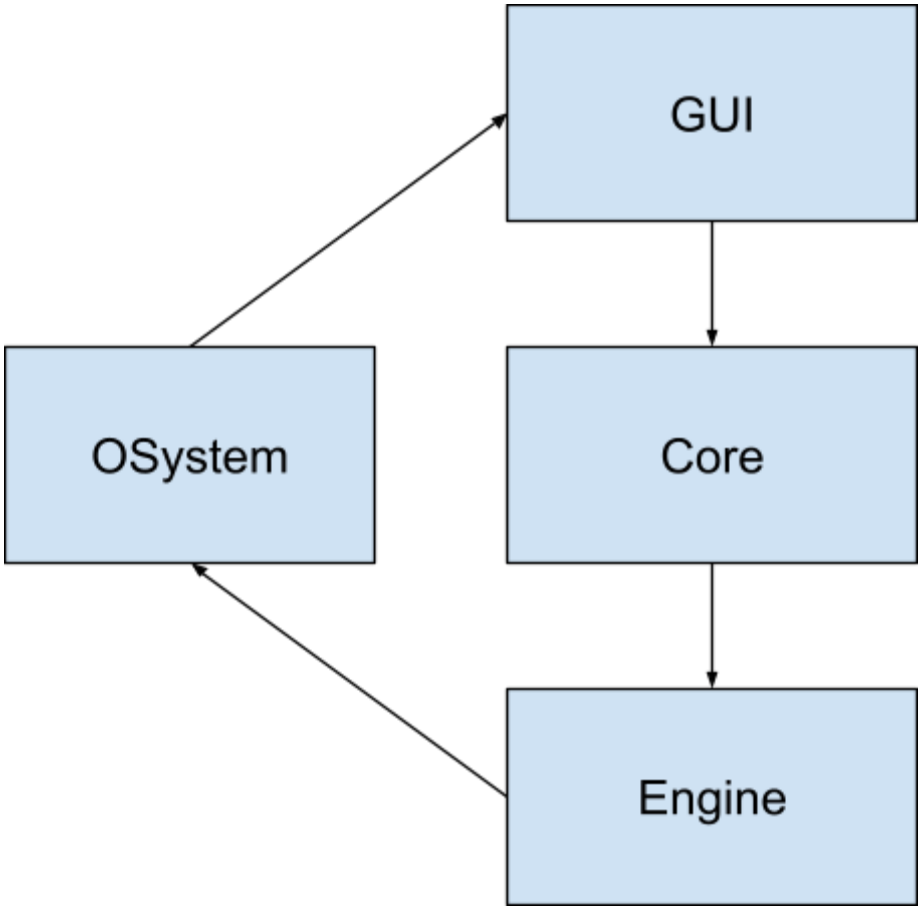


Figure 1. Box-and-arrow diagram of conceptual architecture

Interpreter Pattern

At the core of ScummVM is the interpreter architectural style. Each supported game engine in ScummVM acts as an interpreter for the game scripts of that particular game or set of games [3][6]. The engines parse and execute game scripts, managing game logic, events, and interactions.

This approach allows ScummVM to reimplement the functionality of original game engines without using any of the original executable code, ensuring legality and compatibility across platforms [4][5].

Key Applications:

1. **SCUMM Engine**
    - Interprets SCUMM bytecode used in many LucasArts adventure games [3][4].

    - Supports different versions of SCUMM scripts, handling variations in opcodes and behaviors [3][6].
2. **AGI and SCI Engines**
    - **AGI (Adventure Game Interpreter)** engine interprets AGI logic scripts used in early Sierra games [3][4].
    - **SCI (Sierra's Creative Interpreter)** engine handles more advanced scripts from later Sierra titles [3][4].
3. **Other Engines**
    - Engines like **Kyra** (for The Legend of Kyrandia series) interpret game-specific scripting languages [3][4].
    - Each engine implements an interpreter tailored to its game's scripting language [3][6].

Game engines use a loop that reads opcodes or script commands from the game data. Each opcode is interpreted, and the corresponding action is executed, such as updating game state, triggering animations, or playing sound effects [3][6][14][15]. Interpreters manage game logic without relying on the original executables, ensuring legality and cross-platform compatibility [4][5].

Plug-in Architecture

ScummVM employs a plug-in architecture to manage its support for multiple game engines. Each game engine is implemented as a separate module or plug-in that can be loaded or unloaded dynamically [5][6]. This modularity allows developers to add new engines without modifying the core of ScummVM [5][6].

Key Components:

1. **Engine Plug-ins**
    - Provide methods for detecting supported games, initializing the engine, and handling game execution [5][6].
2. **Plugin Manager**
    - Manages the loading and initialization of engine plug-ins [5][6].
    - Determines which engine to use based on the game data provided [4][5][6].

Implementation:

- **Registration:**
    - Engines are registered using macros or factory methods, enabling ScummVM to recognize and load them [5][6].
- **Detection:**

    - When a user adds a game, ScummVM scans the data files and uses engine plug-ins to detect which engine supports the game [4][5][6][8].
- **Loading:**
    - The appropriate engine plug-in is loaded, and control is passed to it to run the game [5][6].

Layered Architecture

ScummVM uses a layered architecture to abstract platform-specific functionalities and provide a consistent interface for game engines [3][6].

Layers:

1. **Application Layer**

    Contains the user interface and core application logic [6][12][13].

    Handles user interactions, game selection, and settings [6][12][13].

2. **Game Engines Layer**

    Contains the various game engines (interpreters) that run the game scripts [3][6].

    Each engine operates independently, following the plug-in pattern [5][6].

3. **Common Services Layer**

    Provides shared services such as file I/O, memory management, graphics rendering, audio playback, and input handling [3][6].

    Includes utility functions and data structures used across engines [3][6].

4. **Platform Abstraction Layer (OSystem)**

    Abstracts system-specific details like graphics rendering, audio output, input handling, and threading [3][6][14][15][16].

    Defines an interface (`OSystem API`) that the upper layers use to interact with the underlying platform [3][6].

5. **Backends Layer**

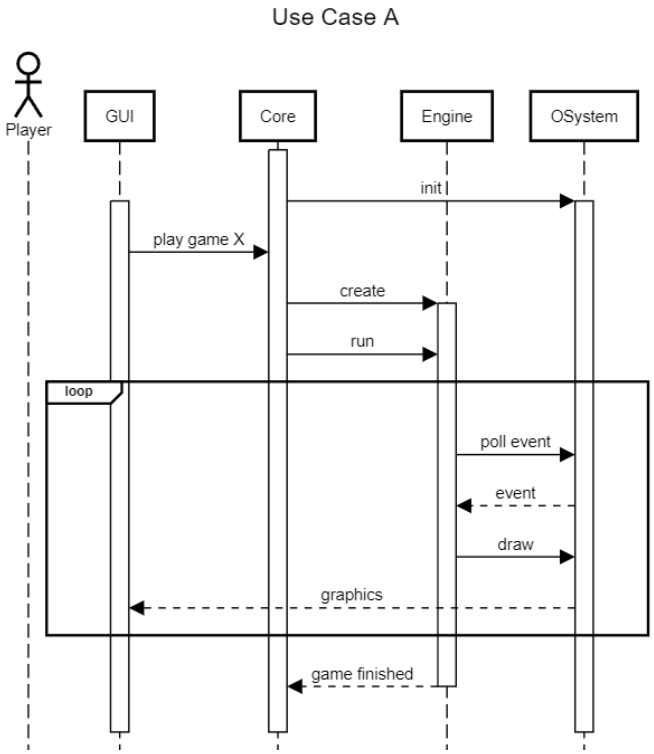
Implements the `OSystem API` for specific platforms (e.g., SDL, Windows API) [3][6][14][15].

Handles the actual communication with the hardware and operating system [3][6].

6. **Hardware Layer**
    - Represents the underlying hardware and operating system [6][14][15][16].

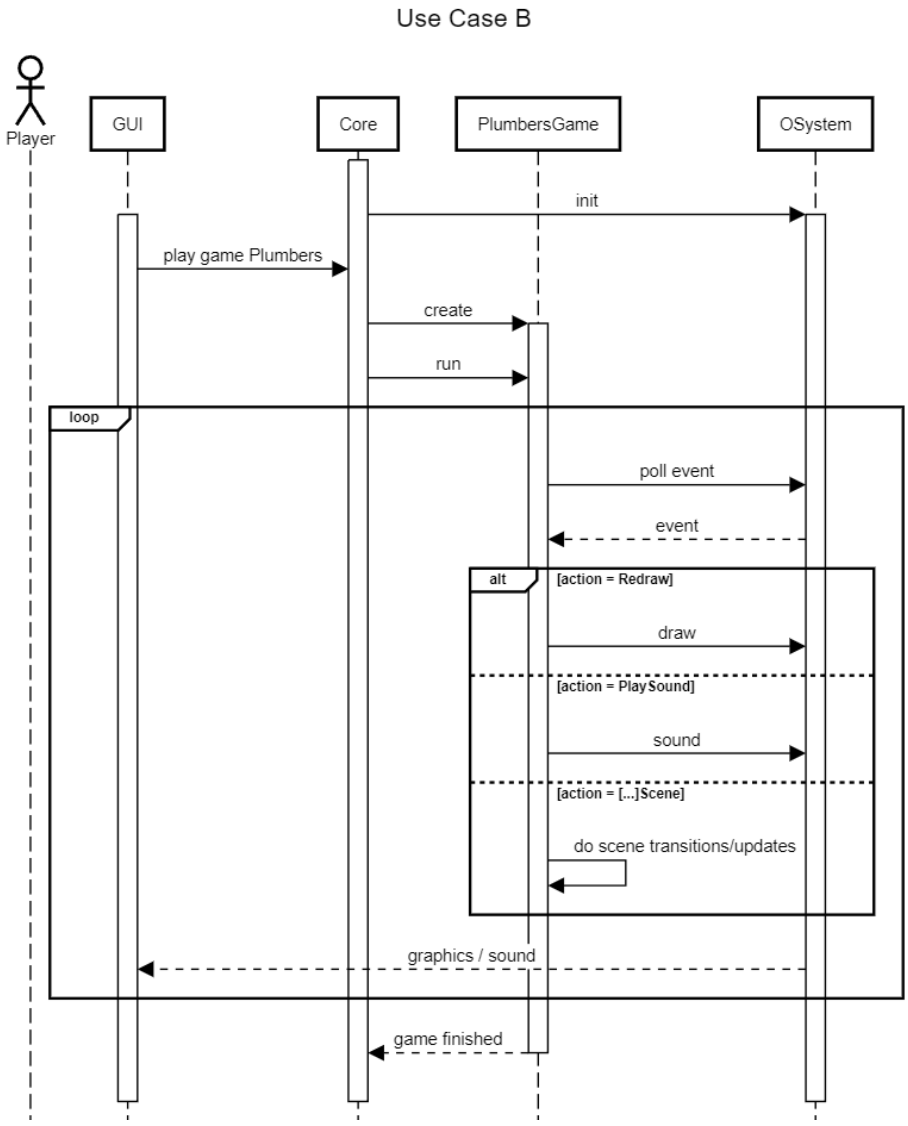
Use Case A:

- The first use case identified is the general sequence for the initialization and running of any game engine



Use Case B:

- The second use case identified is the specific use of a game engine, for this example, Plumbers, as it is the simplest game engine implemented



Component Interactions:

ScummVM's layered architecture facilitates complex interactions between its components while maintaining modularity and portability. Game Engines, sitting at the top layer, interact with Common Services and the Platform Abstraction Layer to perform essential tasks like rendering graphics, playing sounds, and handling input [3][6][14][15][16]. Common Services utilize the Platform Abstraction Layer to access platform-specific features, while the Platform Abstraction Layer communicates with Backends, which in turn interact with the Hardware Layer [3][6]. This structure offers numerous benefits: it ensures portability by keeping upper layers platform-independent, enhances maintainability by minimizing the impact of changes across layers, promotes reusability of common services and utilities across different engines and platforms, and enforces a clear separation of concerns for better code organization [3][6]. However, this architectural approach does come with trade-offs. The abstraction layers may

introduce slight performance overhead, the complexity of managing cross-layer interactions requires careful design to avoid tight coupling, and new developers may face a learning curve in understanding the layered structure [3][6]. Despite these challenges, the benefits of this architecture have been crucial to ScummVM's success in supporting a wide range of games across diverse platforms [1][3][6][7].

History of ScummVM

ScummVM, based on LucasArts' SCUMM engine, began when co-founder Vincent Hamm (aka Yaz0r) developed a SCUMM interpreter for games like Maniac Mansion and Zak McKracken. Though his early progress was limited, he collaborated with Ludde, who was working on a similar project focused on Monkey Island 2. Together, they made significant advances, adding support for Monkey Island 2, Indiana Jones and the Fate of Atlantis, and other SCUMM-based adventure games. Their work involved extensive debugging and improvements, including virtual screen setup, sprite handling, and script compilation.

As the project grew, Yaz0r and Ludde implemented features like sound systems, special save systems, and support for cutscenes. Although they faced challenges with games like Day of the Tentacle and Sam & Max Hit the Road, their efforts made these titles playable. After some time, Yaz0r stepped away from the project, but ScummVM continues to evolve with ongoing updates from the community, ensuring the legacy of classic adventure games lives on.

Event Interface

ScummVM interacts with several external interfaces essential to its operation. The User Interface includes both a Graphical User Interface (GUI) and a command-line interface [6][12][13][17]. The GUI allows users to select and configure games easily, providing an intuitive way to manage settings and launch games [6][12][13]. For advanced users and scripting purposes, the command-line interface offers additional flexibility and control over ScummVM's functionalities [17].

The File System interface is crucial for accessing game data. ScummVM reads game data files from the user's storage, supporting various file formats and directory structures to accommodate the diverse range of games it supports [4][8][32]. This flexibility ensures that users can add games regardless of how their data files are organized [4][8][32].

Graphics Output is handled through rendering graphics to the screen using graphics backends like Simple DirectMedia Layer (SDL) or OpenGL [14][16][20]. ScummVM supports various

resolutions and scaling options, allowing games to be displayed appropriately on modern hardware while preserving the original visual experience [14][16].

For Audio Output, ScummVM outputs sound through the system's audio hardware, supporting multiple audio backends and formats [15][16][20]. This ensures that the game's original audio, including music and sound effects, is faithfully reproduced [15][16].

Input Devices are managed by ScummVM to handle input from keyboards, mice, touchscreens, and game controllers [16][18][19]. The system maps inputs to game-specific actions, enabling users to interact with games as intended, whether through traditional input devices or modern alternatives [16][18][19].

Finally, Configuration Files play a significant role in ScummVM's customization capabilities [9][13][21][22]. The application reads and writes settings from configuration files, allowing users to customize game and system settings according to their preferences [9][13][21][22]. This includes options like key mappings, graphical settings, and audio configurations [13][14][15][19].

These external interfaces are integral to ScummVM's ability to provide a seamless and authentic gaming experience across various platforms and games.

References

[1] ScummVM. (n.d.). ScummVM :: About. Retrieved from https://www.scummvm.org

[2] Wikipedia. (2024). ScummVM. Retrieved from https://en.wikipedia.org/wiki/ScummVM

[3] ScummVM. (2024). ScummVM Documentation. Retrieved from https://docs.scummvm.org/en/v2.8.0/index.html

[4] ScummVM. (2024). ScummVM Documentation: Game Files. Retrieved from https://docs.scummvm.org/en/v2.8.0/use_scummvm/game_files.html

[5] GitHub. (n.d.). ScummVM Repository. Retrieved from https://github.com/scummvm/scummvm

[6] ScummVM. (2024). ScummVM Documentation: How to Use ScummVM. Retrieved from https://docs.scummvm.org/en/v2.8.0/use_scummvm/index.html

[7] ScummVM. (2024). ScummVM Documentation: Install ScummVM. Retrieved from https://docs.scummvm.org/en/v2.8.0/use_scummvm/install_computer.html

[8] ScummVM. (2024). ScummVM Documentation: Add and Play Games. Retrieved from https://docs.scummvm.org/en/v2.8.0/use_scummvm/add_play_games.html

[9] ScummVM. (2024). ScummVM Documentation: Configuration File. Retrieved from https://docs.scummvm.org/en/v2.8.0/advanced_topics/configuration_file.html

[10] ScummVM. (2024). ScummVM Documentation: Graphics Settings. Retrieved from https://docs.scummvm.org/en/v2.8.0/settings/graphics.html

[11] ScummVM. (2024). ScummVM Documentation: Audio Settings. Retrieved from https://docs.scummvm.org/en/v2.8.0/settings/audio.html

[12] ScummVM. (2024). ScummVM Documentation: The Launcher. Retrieved from https://docs.scummvm.org/en/v2.8.0/use_scummvm/the_launcher.html

[13] ScummVM. (2024). ScummVM Documentation: How to Use Settings. Retrieved from https://docs.scummvm.org/en/v2.8.0/use_scummvm/how_to_settings.html

[14] ScummVM. (2024). ScummVM Documentation: Game Settings. Retrieved from https://docs.scummvm.org/en/v2.8.0/settings/game.html

[15] ScummVM. (2024). ScummVM Documentation: Understanding Audio. Retrieved from https://docs.scummvm.org/en/v2.8.0/advanced_topics/understand_audio.html

[16] ScummVM. (2024). ScummVM Documentation: Understanding Graphics. Retrieved from https://docs.scummvm.org/en/v2.8.0/advanced_topics/understand_graphics.html

[17] ScummVM. (2024). ScummVM Documentation: Command Line. Retrieved from https://docs.scummvm.org/en/v2.8.0/advanced_topics/command_line.html

[18] ScummVM. (2024). ScummVM Documentation: Keyboard Shortcuts. Retrieved from https://docs.scummvm.org/en/v2.8.0/use_scummvm/keyboard_shortcuts.html

[19] ScummVM. (2024). ScummVM Documentation: Control Settings. Retrieved from https://docs.scummvm.org/en/v2.8.0/settings/control.html

[20] ScummVM. (2024). ScummVM Documentation: Backend Settings. Retrieved from https://docs.scummvm.org/en/v2.8.0/settings/backend.html

[21] ScummVM. (2024). ScummVM Documentation: GUI Settings. Retrieved from https://docs.scummvm.org/en/v2.8.0/settings/gui.html

[22] ScummVM. (2024). ScummVM Documentation: Misc Settings. Retrieved from https://docs.scummvm.org/en/v2.8.0/settings/misc.html

[23] ScummVM. (2024). ScummVM Documentation: Paths Settings. Retrieved from https://docs.scummvm.org/en/v2.8.0/settings/paths.html

[24] ScummVM. (2024). ScummVM Documentation: Cloud Settings. Retrieved from https://docs.scummvm.org/en/v2.8.0/settings/cloud.html

[25] ScummVM. (2024). ScummVM Documentation: LAN Settings. Retrieved from https://docs.scummvm.org/en/v2.8.0/settings/lan.html

[26] ScummVM. (2024). ScummVM Documentation: Keymaps Settings. Retrieved from https://docs.scummvm.org/en/v2.8.0/settings/keymaps.html

[27] ScummVM. (2024). ScummVM Documentation: Accessibility Settings. Retrieved from https://docs.scummvm.org/en/v2.8.0/settings/accessibility.html

[28] ScummVM. (2024). ScummVM Documentation: Connect to Cloud Storage. Retrieved from https://docs.scummvm.org/en/v2.8.0/use_scummvm/connect_cloud.html

[29] ScummVM. (2024). ScummVM Documentation: LAN. Retrieved from https://docs.scummvm.org/en/v2.8.0/use_scummvm/LAN.html

[30] ScummVM. (2024). ScummVM Documentation: Taskbar Integration. Retrieved from https://docs.scummvm.org/en/v2.8.0/use_scummvm/taskbar_integration.html

[31] ScummVM. (2024). ScummVM Documentation: Understanding the Search Box. Retrieved from https://docs.scummvm.org/en/v2.8.0/advanced_topics/understand_search_box.html

[32] ScummVM. (2024). ScummVM Documentation: Mac Game Files. Retrieved from https://docs.scummvm.org/en/v2.8.0/use_scummvm/mac_game_files.html

[33] ScummVM. (2024). ScummVM Documentation: Save and Load Games. Retrieved from https://docs.scummvm.org/en/v2.8.0/use_scummvm/save_load_games.html