Conceptual Architecture of the ScummVM Engine

Presentation URL: https://www.youtube.com/watch?v=VAsEEQ2vbkc

Group 18

Harry George (Leader) - Sequence diagram 1, Use cases, Title, Authors, Abstract

Deniz Aydin (Presenter) - Introduction/Overview, Lessons learned, Conclusion, Presenter

Kurtis Marinos (Presenter) - Box-and-Line diagram, Data dictionary, Naming conventions, Presenter

Ben Hilderman - Sequence diagram 2, Use cases

Riley Spavor - Architectural styles, Software architecture

Jacob Skiba - Software architecture

Overview

ScummVM Overview:

- Open-source project for running classic adventure and RPG games on modern platforms
- Focus on game preservation and cross-platform compatibility
- Enhances original games with improved graphics scaling and modern control schemes

Primary Goal:

- Unified interface for playing a wide range of classic games from various developers and eras
- Preserves gaming history while improving user experience

Core System Architecture:

- **Design:**
  - Separates game-specific logic from platform-specific implementations
- **Key components:**
  - Core Subsystem
  - Individual Game Engines
  - Platform Ports
  - Support Modules
- Supports over 250 games across 50+ game engines
- Ensures a consistent user experience across platforms (desktop, mobile, consoles)

Architecture Focus:

- System architecture: major components and their interactions
- Control and data flow
- Concurrency management
- External interfaces

Derivations

- **Derivation Process:**
  - Examined project documentation and source code from GitHub repository
  - Analyzed components within ScummVM Core and traced interactions
  - Mapped dependencies between game engines, platform modules, and external interfaces

Key Subsystems Identified:

- **ScummVM Core**
- **Game Engines (e.g., SCUMM, Sword, SCI)**
- **Platform Ports**
- **External Libraries**

Diagram Outputs:

- **Dependency Diagram:** Illustrates component relationships
- **Sequence Diagrams:** Demonstrate game loading and saving use cases

Architectural Style:

- **Layered Architecture with Interpreter Elements:**
  - Core modules coordinate with independent game engines and platform-specific interfaces
  - Supports extensibility: easily add new game engines or platform ports with minimal disruption

System Architecture

ScummVM Architecture Overview:

- **Interpreter, Layered Architecture:**
    - Enhances portability, maintainability
    - Supports multiple game engines for classic games across platforms
    - Combines layered architecture with interpreter style for platform-independent execution

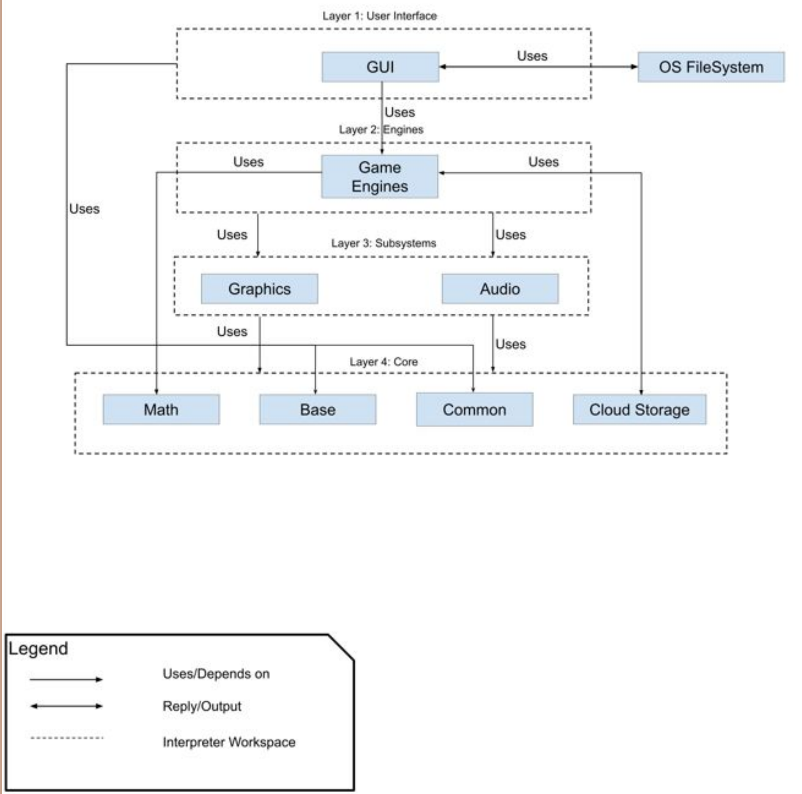
Major Components:

- **Core Subsystem:**
    - Manages memory allocation, file I/O, configuration settings
    - Ensures platform independence by interfacing with backend code
- **Game Engines (e.g., SCUMM, SCI, GrimE):**
    - Interpret game scripts, manage gameplay logic, and handle user inputs
    - Treated as plugins for easy system extension
- **Platform Ports:**
    - Ensures cross-platform functionality via libraries like SDL (for file I/O, graphics, and audio)
- **Support Modules:**
    - Handles audio formats, graphics rendering, and system compatibility

Subsystem/Component Breakdown

- **Engines (SCUMM, SCI):**
  - Interpret game logic, process player inputs, manage game states
- **Common Code:**
  - Shared utilities for file handling, memory management, encoding
- **Audio Subsystem:**
  - Supports sound mixing, MIDI playback, digital sound effects across platforms
- **Graphics Subsystem:**
  - Abstracted rendering using SDL for consistent visuals
- **GUI Subsystem:**
  - Manages user interfaces like menus and settings
- **Math Subsystem:**
  - Performs essential calculations (vector, coordinate transformations)
- **Base Subsystem:**
  - Initializes system, loads plugins, manages memory, and input processing

Box-and-Line Diagram



Subsystem/Component Breakdown

Control and Data Flow:

- **Layered Structure:**
  - User Interface Layer captures inputs, Engines Layer interprets game logic
  - Subsystems handle resources (sprites, audio), updates game state, sends to UI Layer for rendering

Concurrency and Thread Management:

- **Primarily Single-Threaded Model:**
  - For linear structure of older games
  - Asynchronous handling for specific tasks (audio streaming, background resource loading)

Dependencies Between Subsystems:

- **Game Engines:** Rely on Common Code for memory and file handling
- **Common Subsystem:** Provides shared services (file, memory management) to all engines
- **Audio Subsystem:** Synchronizes audio with game events across platforms
- **Graphics Subsystem:** Renders visuals and interacts with game logic
- **GUI Subsystem:** Manages user interface and game settings
- **Math Subsystem:** Supports game logic with calculations and transformations
- **Base Subsystem:** Handles system initialization, plugin management, and cross-platform operations

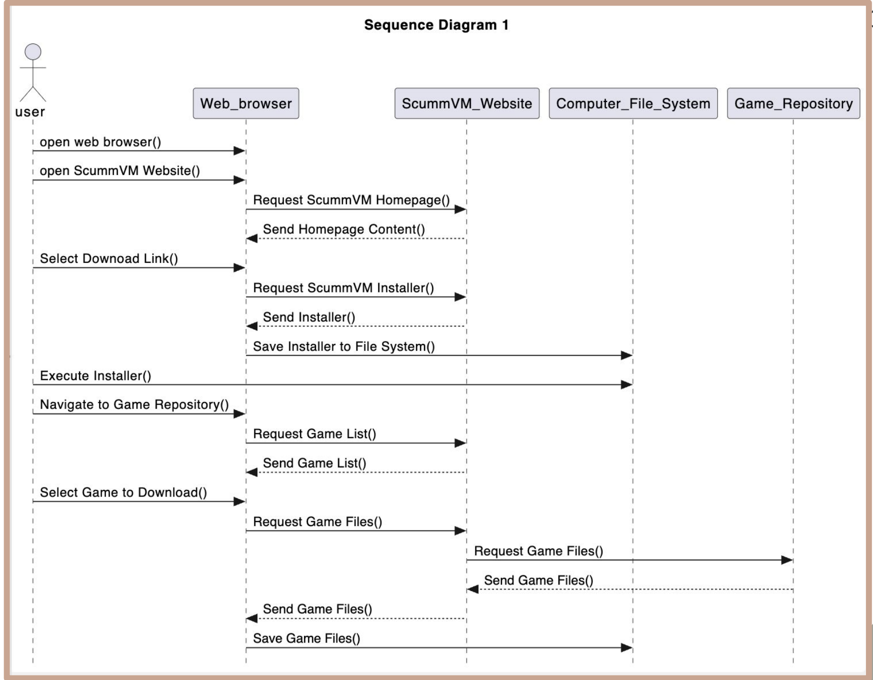
External Interfaces

1. **User Input Interfaces:**
    - Captures inputs from devices like keyboards, mice, game controllers
    - Processed via Core Subsystem, translated into commands for game engines
    - Uses libraries like SDL to ensure cross-platform compatibility
2. **File System Interfaces:**
    - Loads game assets, saves game states, and manages config files
    - Interacts with platform-specific file systems via Data Tier abstraction
    - Handles user preferences through configuration files
3. **Graphics and Audio Interfaces:**
    - Uses libraries like SDL, OpenGL, ALSA for rendering graphics and playing audio
    - Supports various resolutions, aspect ratios, and audio formats
    - Managed by the Presentation Tier for flexibility across devices
4. **Game Engine Interfaces:**
    - Game engines interpret scripts, control game flow, and interact with Core Subsystem
    - Standardized interfaces allow easy addition of new game engines
5. **Platform Port Interfaces:**
    - Platform-specific backends handle rendering, input, and audio playback
    - Translates platform-specific calls (e.g., DirectSound, PulseAudio) into standard ScummVM interfaces
    - Enables cross-platform compatibility (e.g., desktops, consoles)
6. **Network Interfaces (Optional):**
    - Used for downloading assets, updates, and documentation
    - Managed by Core Subsystem using libraries like libcurl
    - Optional and can be disabled for offline use, preserving portability

Use-Case I

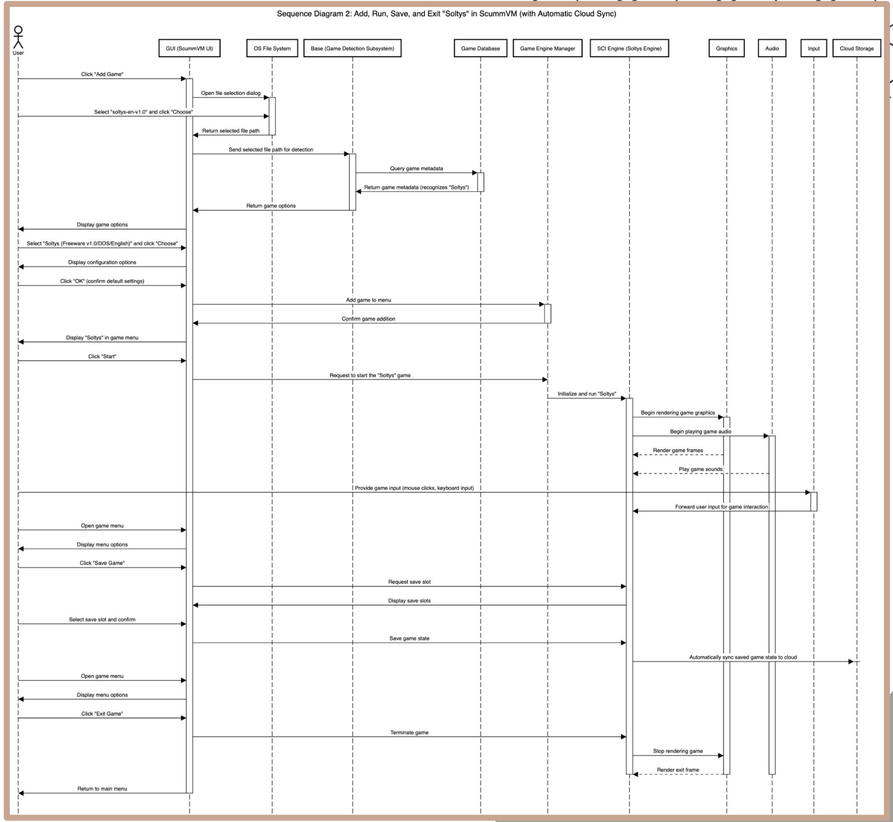
Downloading and Installing ScummVM

1. **Navigating to the ScummVM Website:**
   - User opens a web browser and visits the ScummVM website
   - The website handles HTTP requests, providing access to the downloadable installer
2. **Downloading the Installer:**
   - Browser downloads the ScummVM installer
   - Installer is saved to the local file system
3. **Installing ScummVM:**
   - User executes the installer, activating the ScummVM Core component
   - Core sets up the necessary environment and configures software on the local system
4. **Accessing the Game Repository:**
   - User navigates to the Game Repository via a web browser
   - ScummVM Core interacts with the repository to request and display supported games
5. **Downloading Game Files:**
   - User selects a game; ScummVM sends a request to the Game Repository
   - Game files are retrieved over the network and saved to the local file system



Use-Case II

1. **Adding the Game:**
   - User clicks "Add Game" in the ScummVM UI
   - **Base (Game Detection Subsystem)** checks selected files for known game data
   - Base returns metadata for "Soltys" to the user
2. **Confirming Game Options:**
   - User confirms game settings
   - **Game Engine Manager** adds "Soltys" to the ScummVM game menu
3. **Starting the Game:**
   - User clicks "Start" to launch the game
   - **SCI Engine** is initialized to run the game
   - SCI Engine interacts with **Graphics and Audio subsystems** for visuals and sound
4. **Gameplay:**
   - User can open the in-game menu to load or save the game state
   - Clicking "Save Game" saves the game state locally via the **SCI Engine**
   - Game state is synced to the cloud via the **Cloud Storage subsystem**
5. **Exiting the Game:**
   - User opens the in-game menu and selects "Exit Game"
   - SCI Engine stops, graphics rendering and audio playback halt
   - User is returned to the ScummVM main menu



Conclusion/Lessons Learned

Design strengths:

- High portability.
- Flexibility.
- Strong separation of concerns between game-specific and platform-specific components.

Challenges:

- Complexity can lead to maintenance issues.
- Adding new game engines or optimizing for modern hardware can be difficult.

Proposed future enhancements:

- Add support for newer game engines.
- Implement a more robust multi-threading model for resource-intensive titles.
- Improve the user interface to support high-resolution displays.

Outcome: Enhancements will help ScummVM remain a reliable tool for game preservation and accessibility.

Thank You

References

1. *ScummVM Documentation*. ScummVM, https://docs.scummvm.org.
2. *Developer Central*. ScummVM Wiki, https://wiki.scummvm.org/index.php?title=Developer_Central.
3. *ScummVM Forums*. ScummVM, 27 Mar. 2009, https://forums.scummvm.org/viewtopic.php?t=7886.