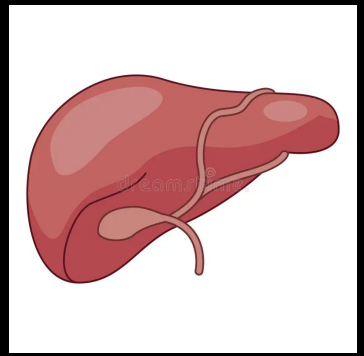
Bile Bytes

ScummVM Architecture



Video presentation link

Queen's University

Bile Bytes: Contributions

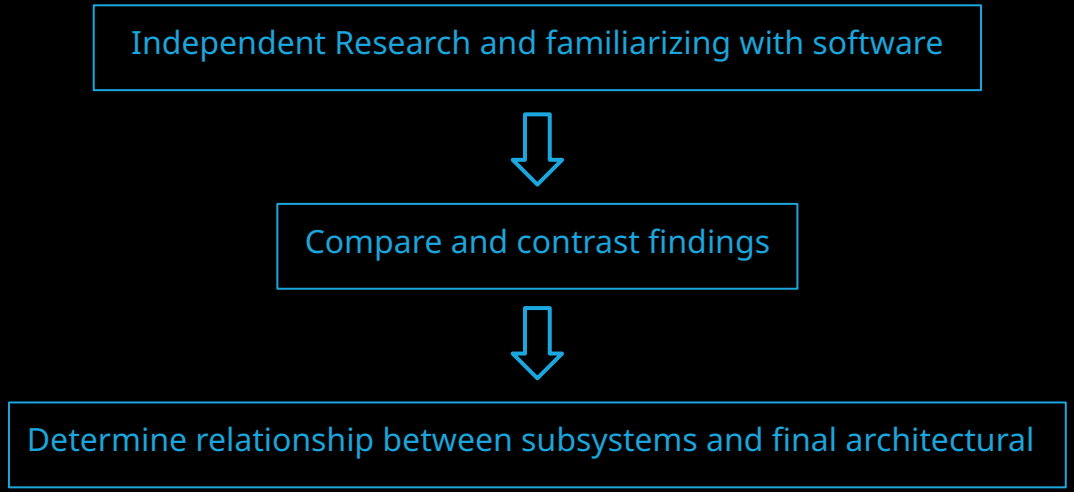
- John Li (Group Leader)
  - Game engines subsystem
  - Concurrency
- Ewan Byrne
  - Use cases and sequence diagrams
  - Derivation process
  - Conceptual architecture (diagram)
- Brandon Kim (Presenter)
  - Use cases and sequence diagrams
  - Derivation process
- Norah Jurdjevic
  - Other subsystems
  - Abstract
  - Conclusion
- Gavin Yan
  - Conceptual architecture
  - Division of responsibilities among developers
- Owen Sawler (Presenter)
  - Introduction
  - Evolution

Introduction & Overview

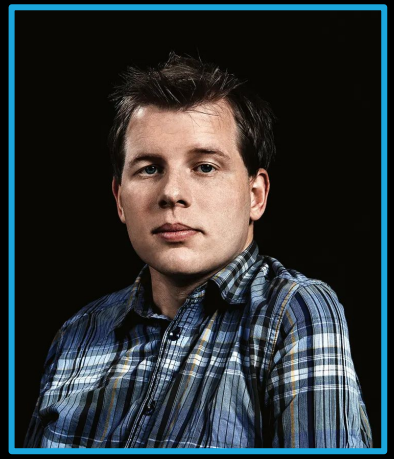
- What is ScummVM?
  - ScummVM: Script Creating Utility for Maniac Mansion Virtual Machine.
  - Interprets old point-and-click adventure games to run on modern systems.
  - Open source and modular nature allows for developers to create their own engines within ScummVM.
  - Prevents classic games from becoming obsolete as a result of hardware advancements.
- Overview
  - Discussion of ScummVM's architectural styles
  - Subsystems of ScummVM, descriptions of the components and their interactions
  - Relevant use cases that highlight the architectural style and interactions between key components
  - Evolution of the system

The Secret of Monkey Island

Derivation Process



Evolution



SCUMM engine created by LucasArts. First used in Maniac Mansion.

1987

2001

ScummVM project launched by Ludvig Strigeus with the goal to rebuild the SCUMM interpreter for modern systems

Added support for engines beyond SCUMM such as AGOS, Sky, Queen, AGI, Cine, Tinsel, SCI, and more.

2002 - 2009

2010 - 2014

Added multi-platform support such as PlayStation 3, IOS, Android, Wii.

Added support for more engines such as Toon, TsAGE, CGE, Dreamweb, NGI, BBVS, and more.

More support for engines such as Sherlock, Lab, Gnap, ADL

2015 - 2020

Present Day

Vast preservation ecosystem that supports 70+ game engines and hundreds of classic titles.

Layered Style

- ScummVM is divided into distinct layers where each builds upon the services of the one below it

UI Layer

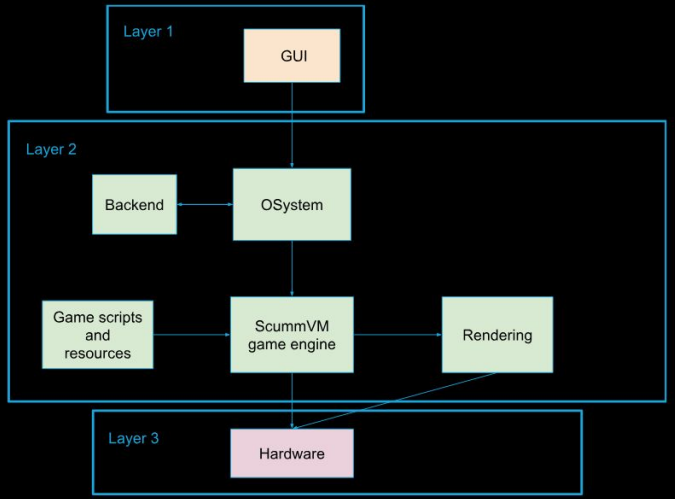
- **Components:** GUI, User Input
- Abstracts game engine complexity
- Handles platform-independent interaction

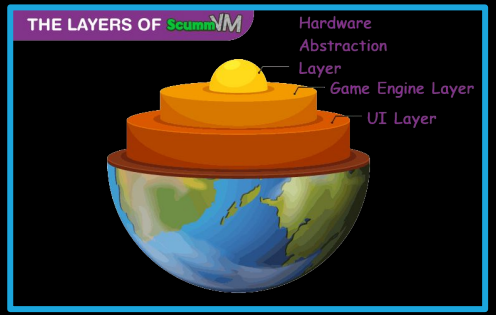
Game Engine Layer

- **Components:** OSystem, Game Engine
- Handles game logic and engine functions
- Interacts with the backend for OS-specific tasks and rendering

Hardware Abstraction Layer

- **Components:** Physical Hardware
- Executes system-level operations



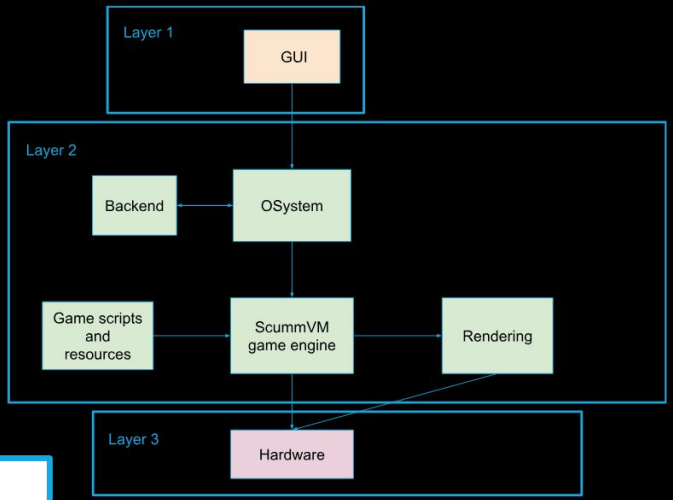


Interpreter Style

- Dynamically reads, parses, and executes game instructions at runtime (instead of pre-compiling them)

Components:

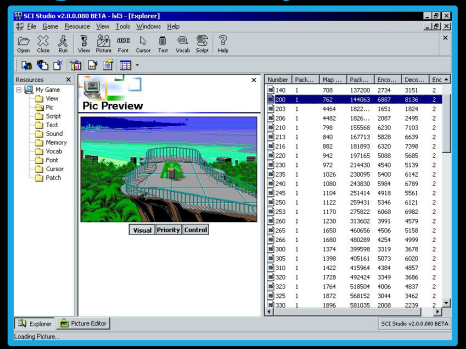
- **Game Engine:** Acts as the interpreter, reading and executing game scripts
- **Game Scripts:** Provide logic, flow, and events (e.g., displaying images, playing sounds)
- **Rendering & Backend:** Handle OS-specific tasks and graphical output

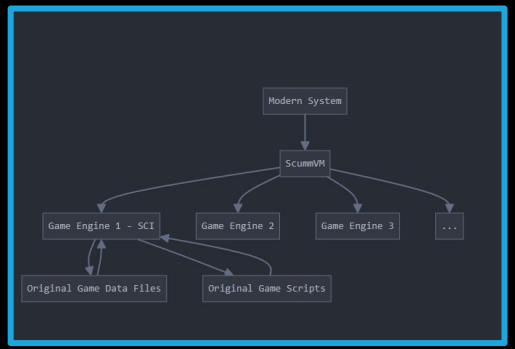


Subsystems

Game Engines

- Each game engine is designed to replicate the functionality of the original engines (e.g, SCI)
- Engines are responsible for interpreting the game's original data files and scripts, translating them into instructions modern systems understand
- Modular subsystem architecture of ScummVM's game engines ensure each engine can operate independently of each other
- Developers are able to contribute new engines for currently unsupported games without disrupting existing functionality





Subsystems

OSystem API

- Responsible for defining which available features a game can use
- Handles graphics, timing, hardware inputs, keybindings and thread management
- Serves as a bridge between the platform implementation in the backend and the frontend

Backends

- Responsible for the implementation of the OSystem API
- Provides a video surface which ScummVM can draw in, methods to create timers and handle user input, and controls for CD playback and other sounds
- Backends are organized into a number of directories which contain information and methods accessed by the OSystem API on behalf of the game's frontend

Concurrency

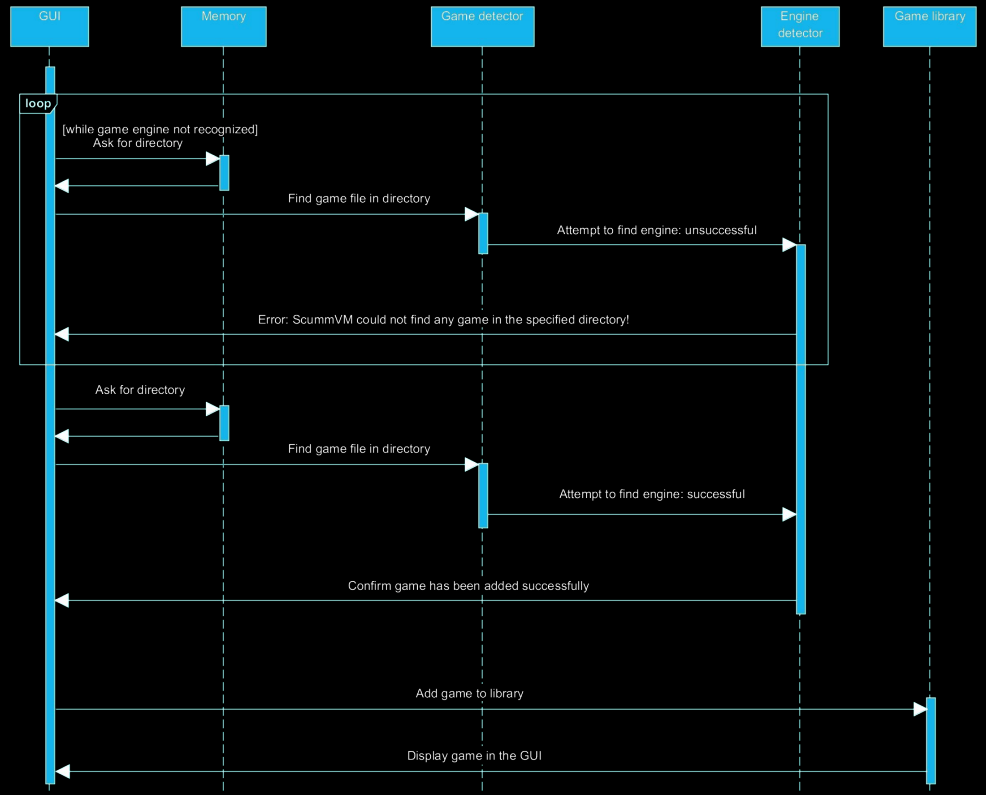
ScummVM was designed to emulate classic games from a single-threaded era, so ScummVM was not designed for and does not rely heavily on concurrency.

Some elements of ScummVM like audio processing may involve concurrency, such as handling audio streaming in separate threads or audio buffering. However, the extent that ScummVM utilizes multithreading is limited.

Use Cases

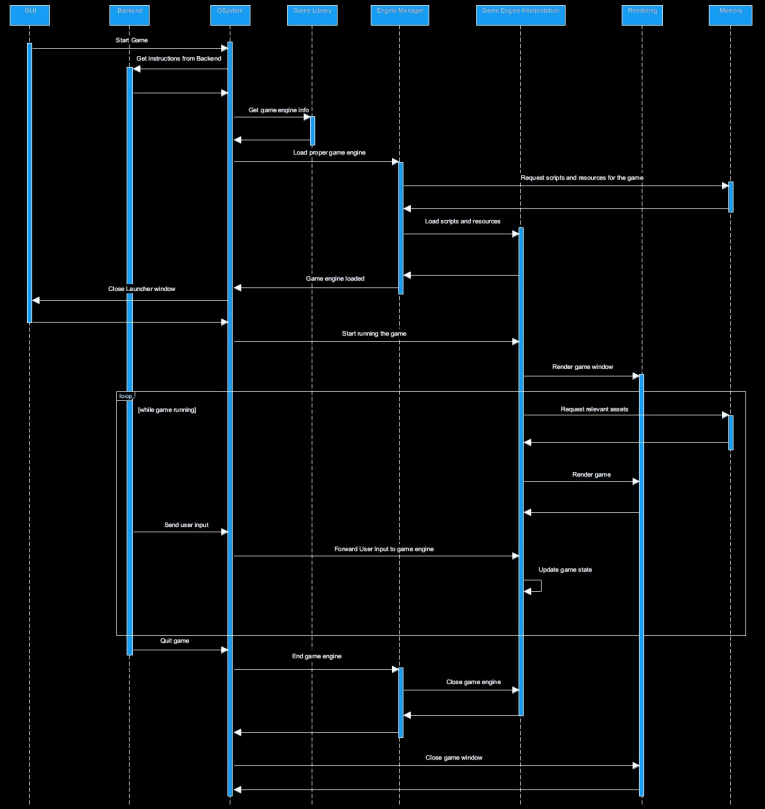
- Adding a game to ScummVM
- Playing a game through ScummVM
- Saving a game

Use Case: Adding a Game to ScummVM



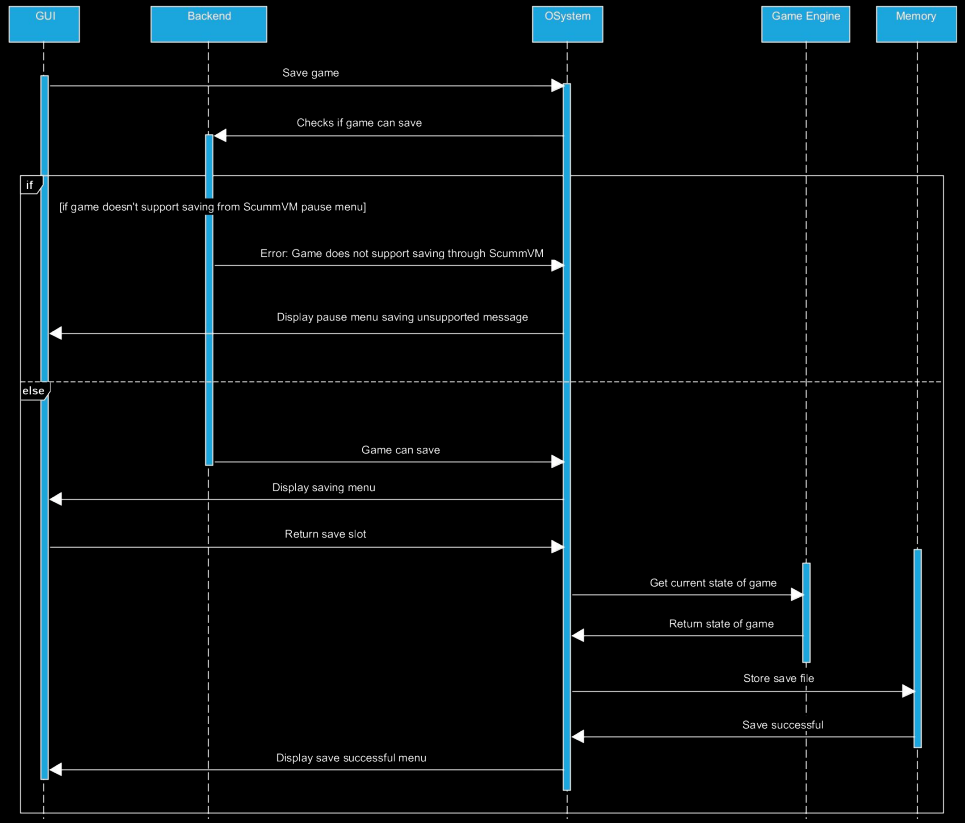
- Infrequent but necessary use case
- User interacts with GUI
- ScummVM detects the game and the corresponding engine, then stores it in the library

Use Case: Playing a Game Though ScummVM



- The core of the software
- Very common use case
- OSystem plays a central role
- Shows interactions between key components (Game Engine, OSystem, Backend)

Use Case: Saving a Game



- Useful feature
- Highlights OSystem being the sole communicator with Backend

Division of Responsibilities Among Developers

- ScummVM allows classic games to run on modern systems without changes from original developers
- Responsibility can be separated into two teams:
  - Game Engine rewriters
  - ScummVM developers
- Game engine rewriters reverse engineer and adapt older game engines
- ScummVM developers integrate these engines into the platform, ensuring compatibility
- Both teams work mostly independently, communicating on changes to the interface

Lessons Learned

- Summary of components is very useful for developing a surface-level understanding
- Performing research independently allowed maximum generation of ideas, while minimizing confusion
- Working on separate sections communicating was a good way of ensuring that the architecture remained cohesive

Conclusion

ScummVM’s architectural style uses a combination of layered and interpreter styles. The separation of the user interface, game engine and hardware and resources into separate layers improves modularity and maintainability, while the interpreter allows ScummVM to support a variety of game engines.

Our research discusses the platform’s three primary subsystems, evolution and limited concurrency. Through the illustration of several key use cases, the interaction between key components and overall functionality of ScummVM is further illuminated.

ScummVM’s open source structure ensures its continued evolution as users are invited to make changes driven by their needs. It provides a valuable service in a rapidly modernizing world, and will likely continue to grow in popularity as game technologies continue to advance.