Bile Bytes

ScummVM Concrete Architecture

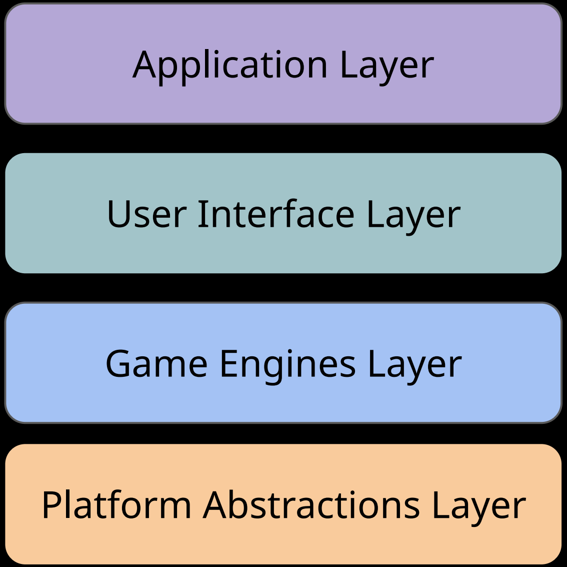
https://drive.google.com/file/d/15glZ6RTu7kyJ9DMq_jQpUZDmkT5hiUOZ/view?usp=drive_link

Bile Bytes: Contributions

- John Li (Group Leader)
  - Dependency discrepancies
  - Use case 2 derivation
- Ewan Byrne
  - Use cases and sequence diagrams
  - Conceptual architecture revision
- Brandon Kim (Presenter)
  - Game Engine subsystem reflexion analysis
- Norah Jurdjevic
  - Second level subsystem analysis
  - Abstract
  - Conclusion
- Gavin Yan
  - Concrete architecture and derivation process
  - Interactions between subsystems
- Owen Sawler (Presenter)
  - Introduction
  - ScummVM reflexion analysis

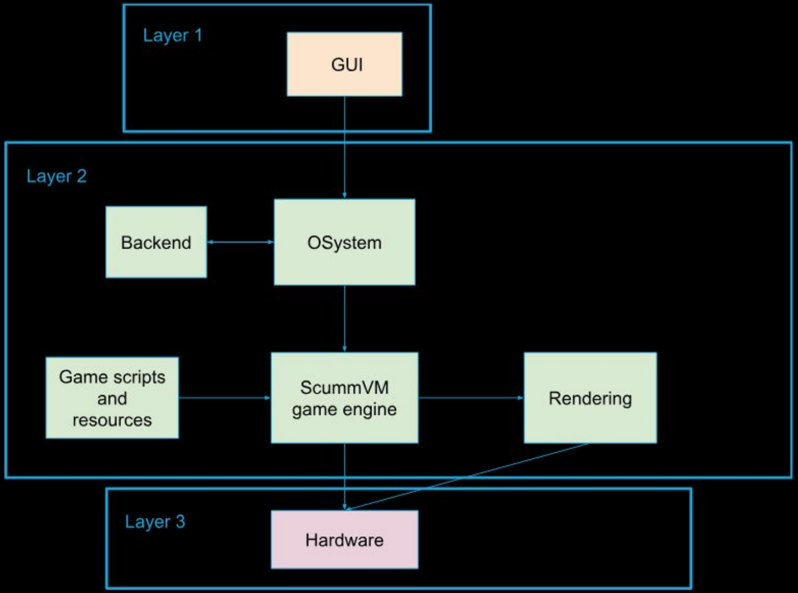
Introduction

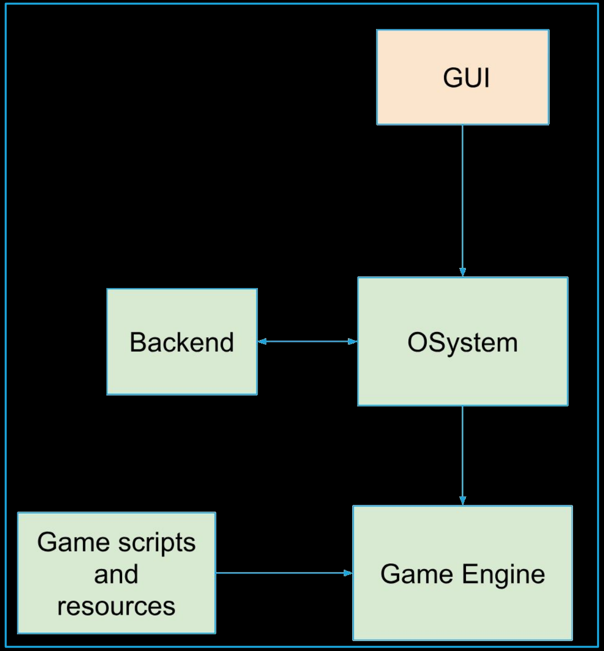
- **Focus:** Continuing with our research on the ScummVM system, we analyzed ScummVM’s concrete architecture, highlighting key differences from its conceptual architecture.
- **Key Insights:** Explores the Game Engines subsystem and its interaction with ScummVM’s architecture.
- **Methodology:** Utilizes the software reflexion framework and sequence diagrams to illustrate use cases and application flow.



Revised conceptual architecture

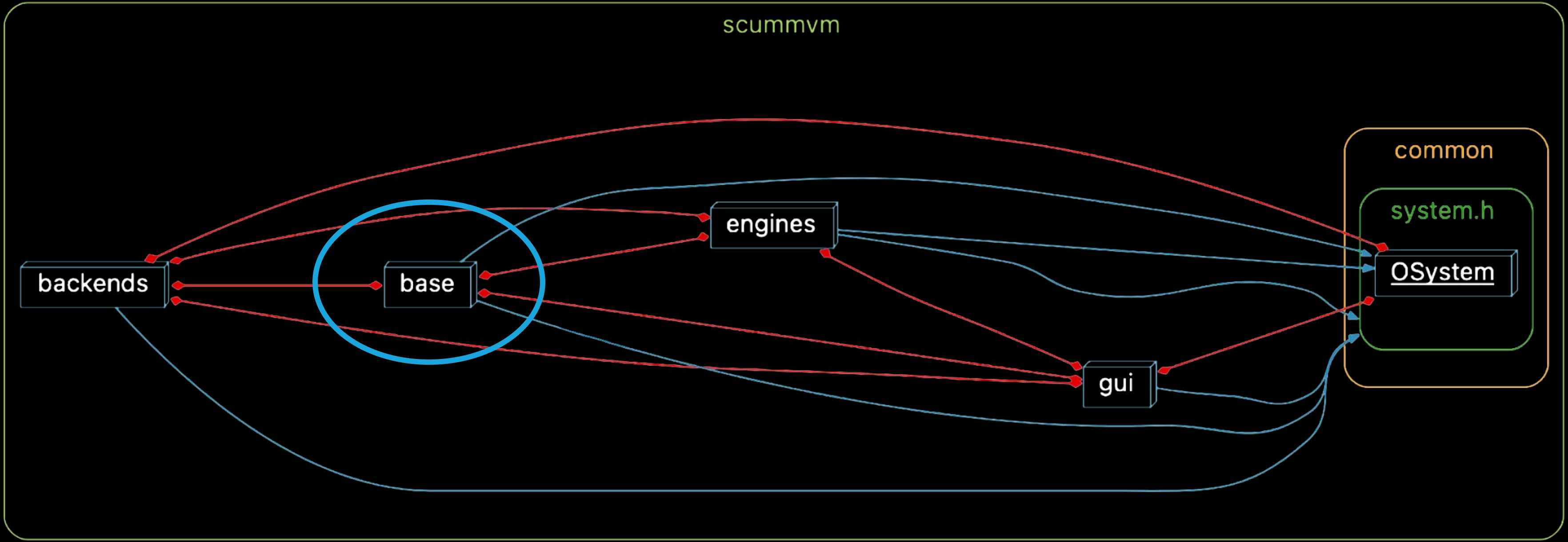
- Changes made to simplify comparisons to dependency graph in Understand
- Hardware and rendering removed, not a part of the software architecture





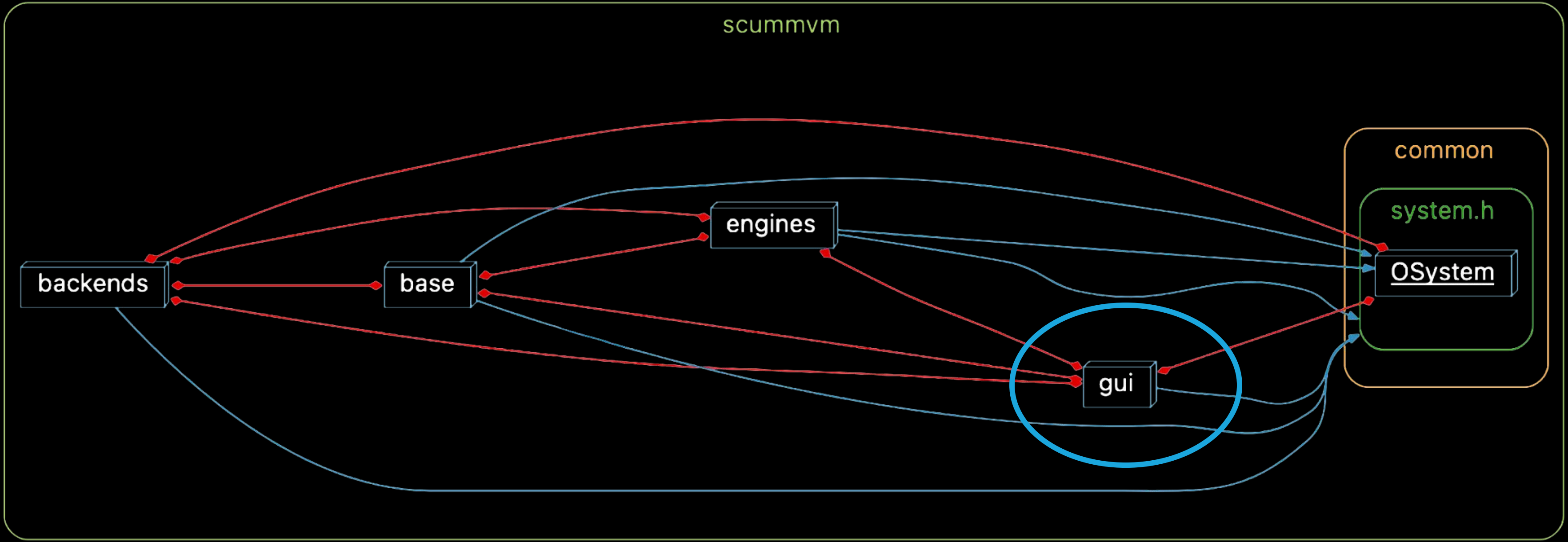
Concrete Architecture

- **Application Layer (base/)**
  - Manages application startup, main loop, and command-line interface, acting as the entry point for the entire program



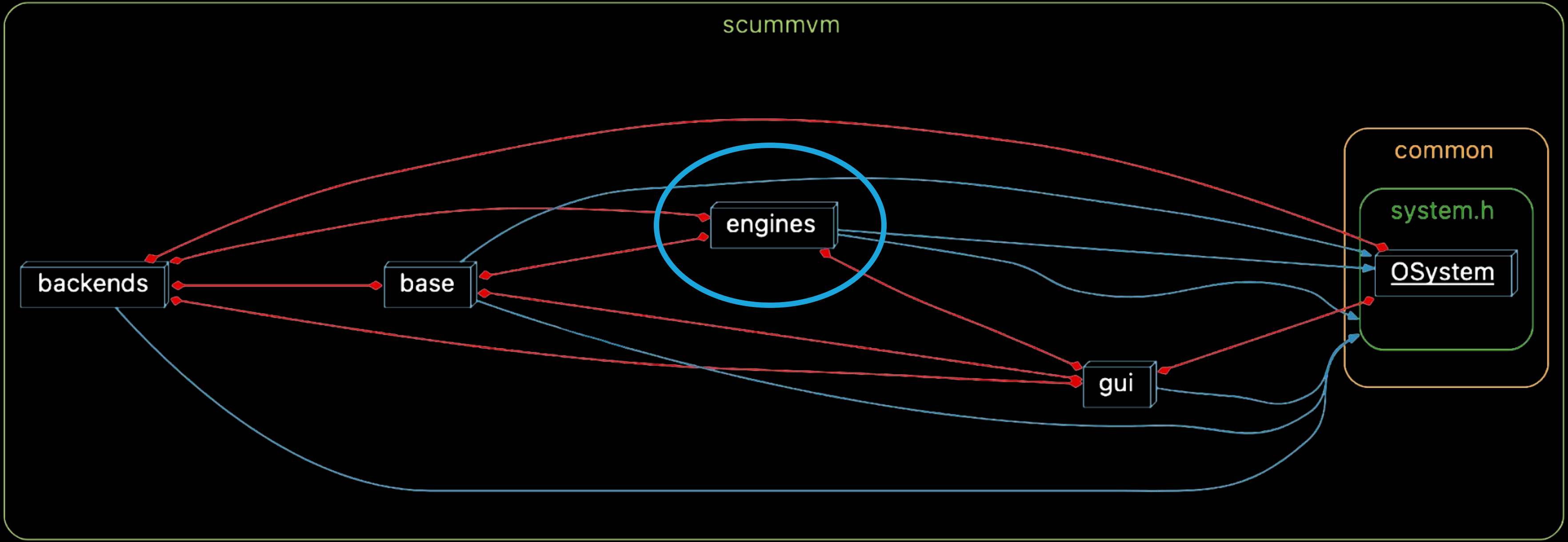
Concrete Architecture

- **User Interface Layer (gui/)**
  - Provides the graphical interface (menus, dialogs) for user interactions, handling game selection, configuration, and input



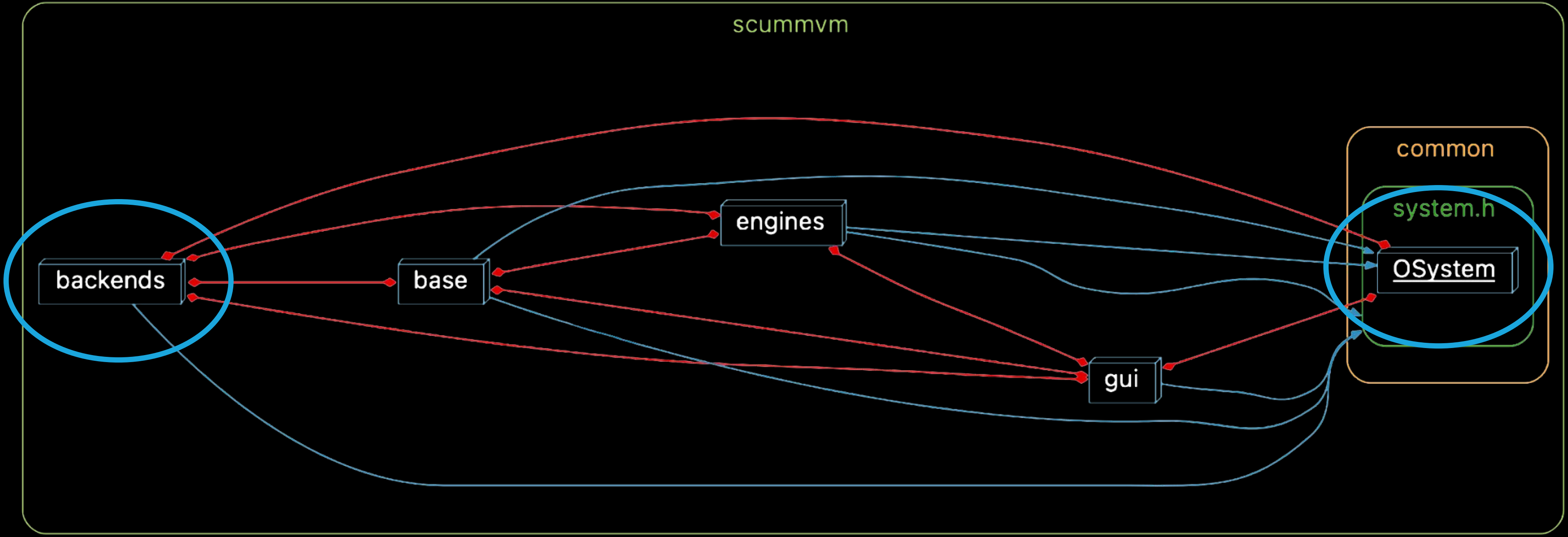
Concrete Architecture

- **Game Engine Layer (engines/)**
  - Hosts individual game engines (e.g., SCUMM, AGI) for interpreting game data and executing gameplay logic



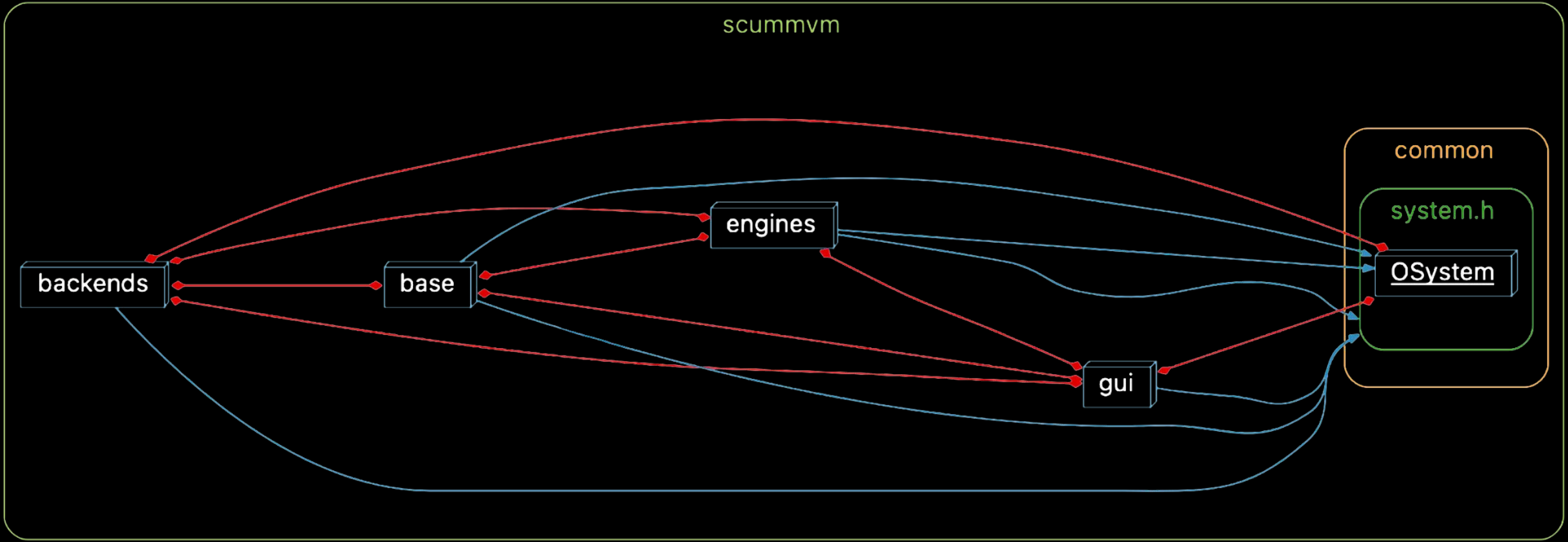
Concrete Architecture

- **Platform Abstraction Layer (common/system.h, backends/)**
    - Abstracts platform-specific details, providing a unified interface for different operating systems



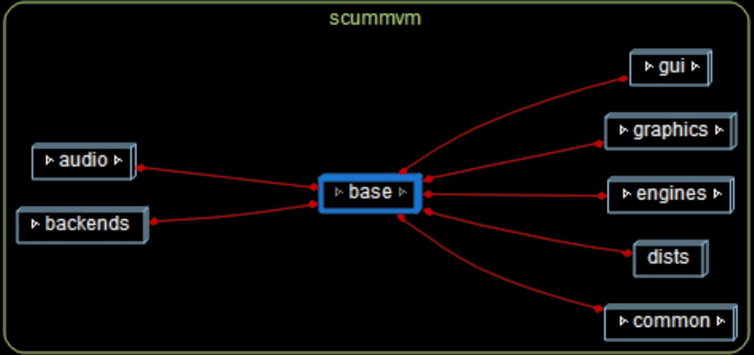
Concrete Architecture

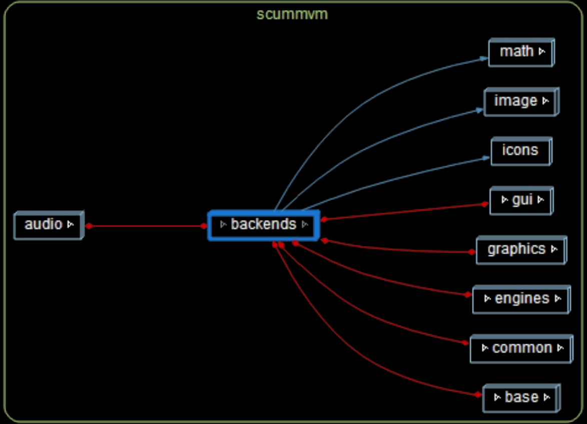
- **Shared Utilities (common/, audio/, image/, video/, math/)**
  - Offer reusable components for common functionalities like file handling, graphics, and audio processing, supporting modular and consistent code across the system



High-Level Architecture Reflexion Analysis

- Divergence 1: Base Component Emergence:
  - Appeared at the top of the Application layer in the concrete architecture
  - Acts as the entry point for the program and central hub, overseeing interactions amongst subsystems.
- Divergence 2: Repositioning of the User Interface layer
  - In the conceptual architecture, was placed at the topmost layer. In practice, is placed below the Application Layer containing the base/ component.
  - Convergence: The system is still divided into distinct layers with defined roles, even though the boundaries between layers have been shifted.
- Divergence 3: Backend and OSystem Reorganization
  - Previously, the Backends and OSystem were middle-layer components, mediating between the GUI and hardware components.
  - In reality, they form the Platform Abstraction layer, directly interacting with gui/ and engine/ components
  - Increases efficiency but introduces tighter coupling between isolated components.
- Unexpected bi-directional dependencies emerged between the GUI and OSystem components.



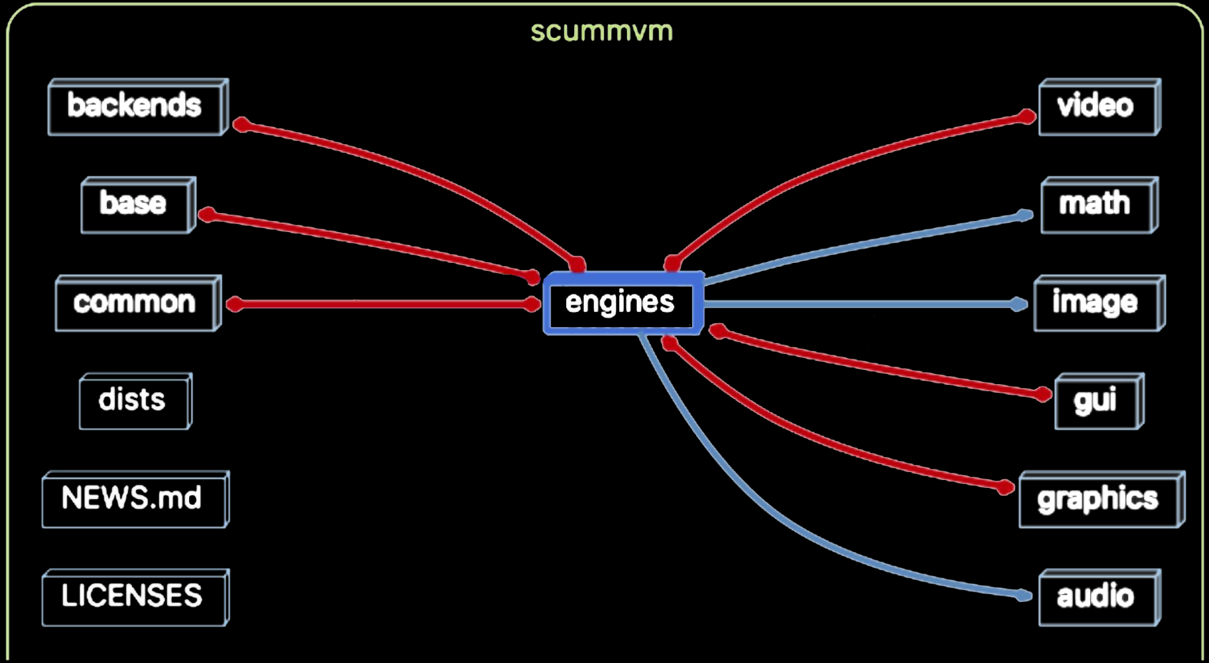


Second Level Subsystem (Game Engines)

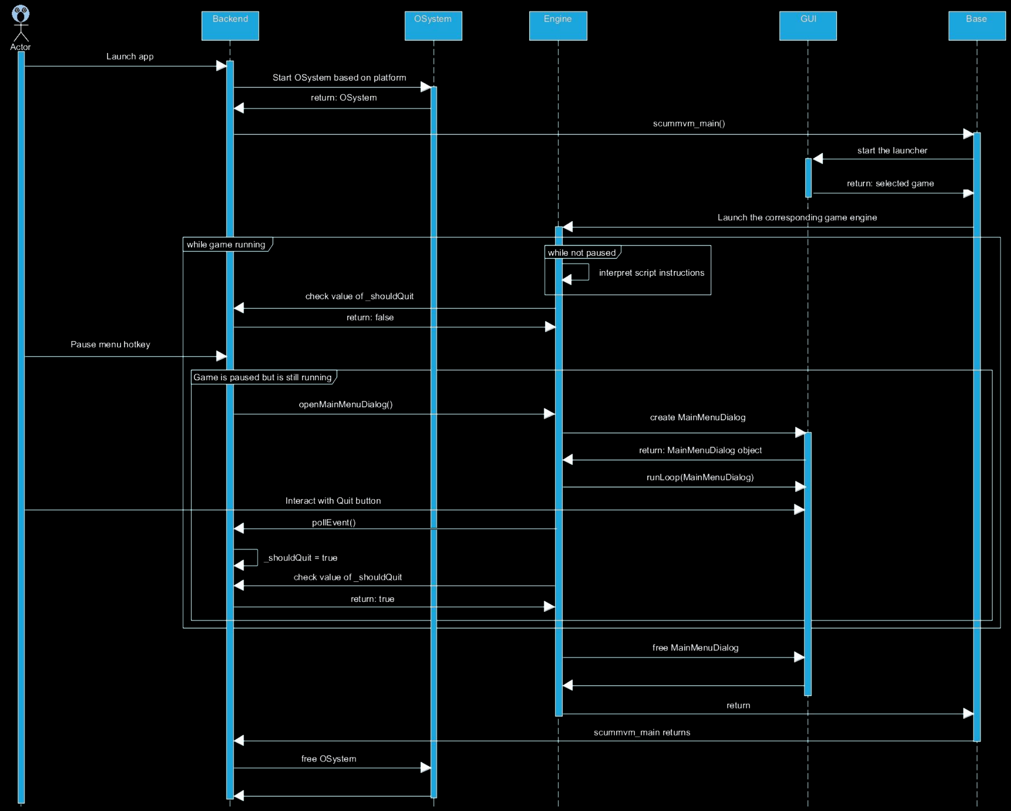
- The Game Engines subsystem encapsulates game-specific logic, file handling and rendering mechanics for emulating original game behaviour.
- It follows the interpreter architectural style, where each ScummVM game engine implementation acts as an interpreter. Each engine interprets high level game scripts into specific low level instructions for the system to execute.
- Concrete implementations for specific engines extend the base class to define game-specific behaviors and interactions.
- These engines act as isolated modules which use common libraries for cross-platform support and multimedia handling, allowing different engines to reuse the same utilities while implementing distinct game logic.

Reflexion Analysis

- The engine subsystem does not have a dependency with the OSystem

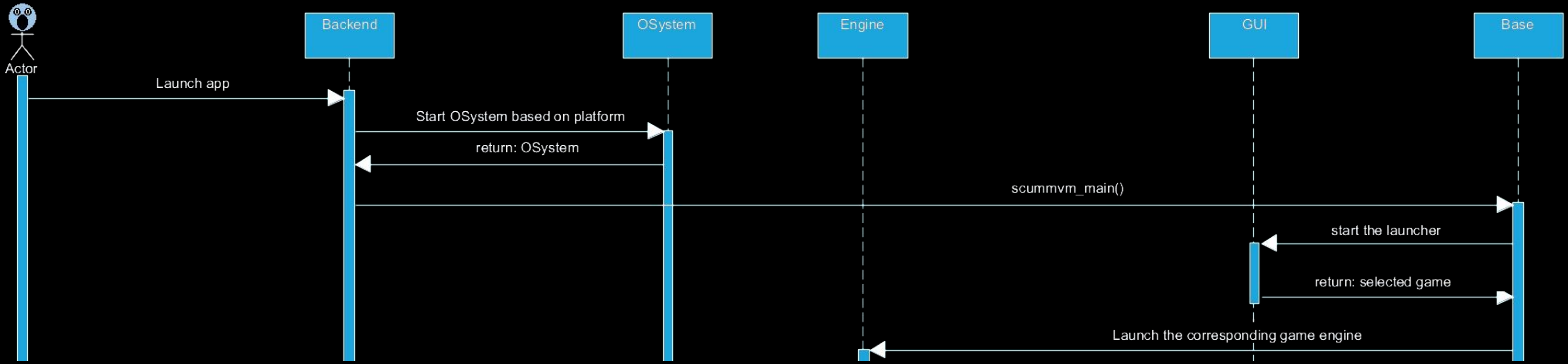


Use case 1: Playing a game

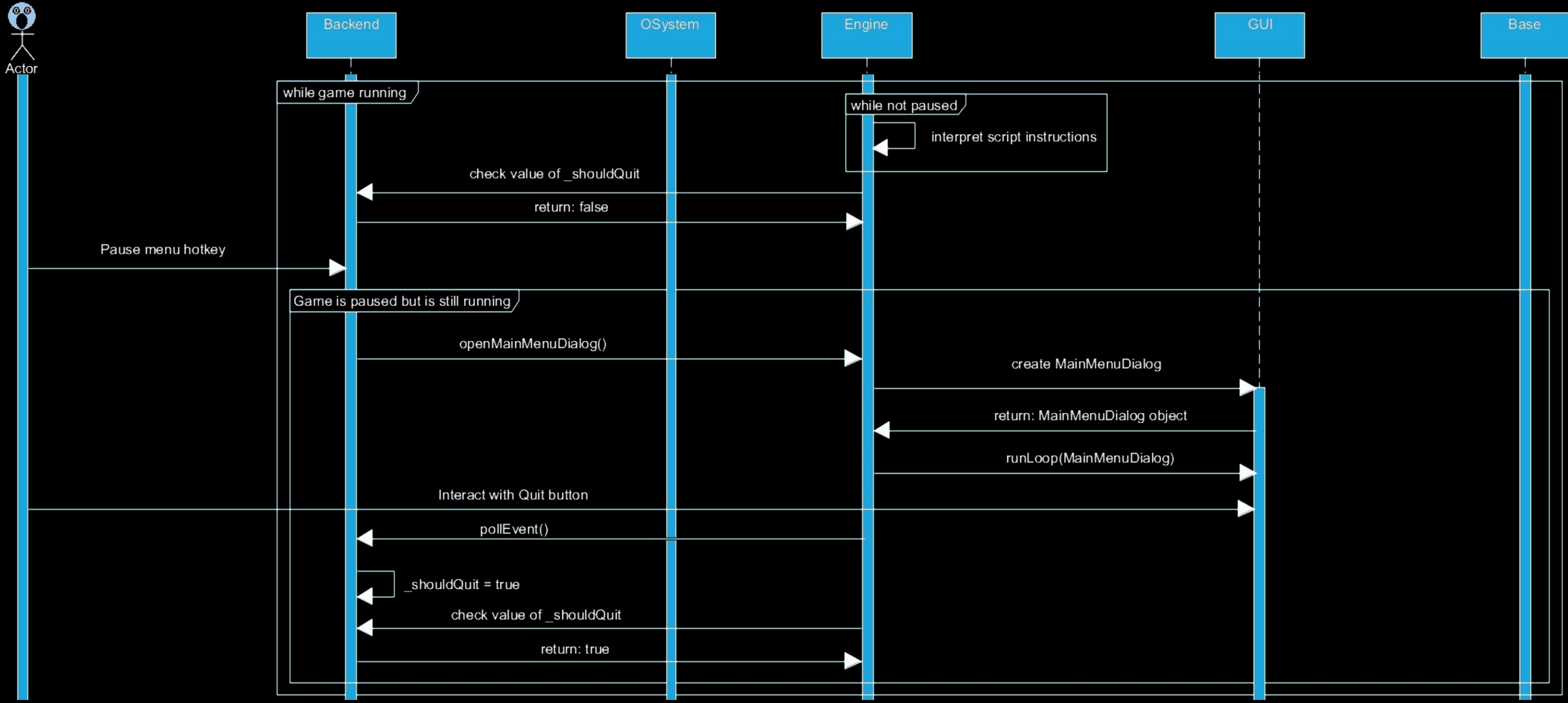


- 3 parts:
  - Starting the game
  - Game Loop
  - Quitting the game

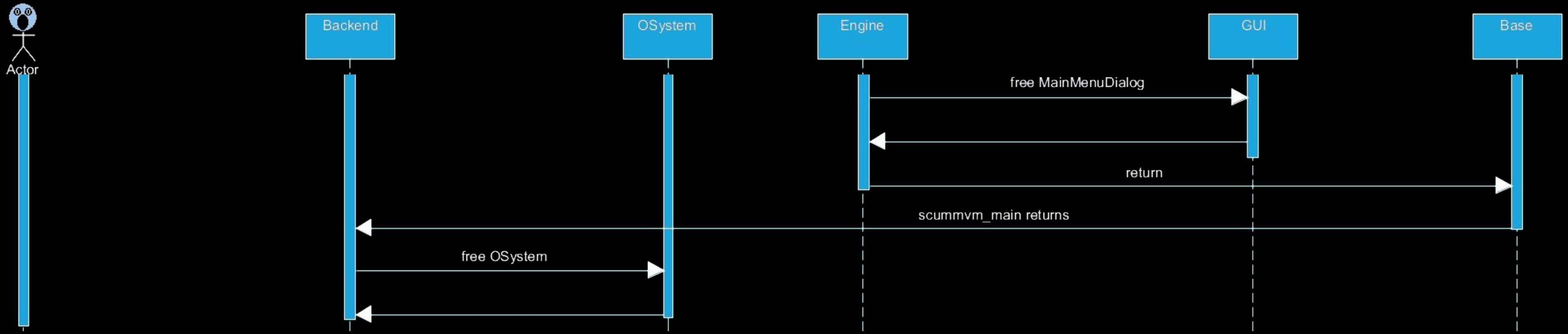
Use case 1: Playing a game - Starting the game



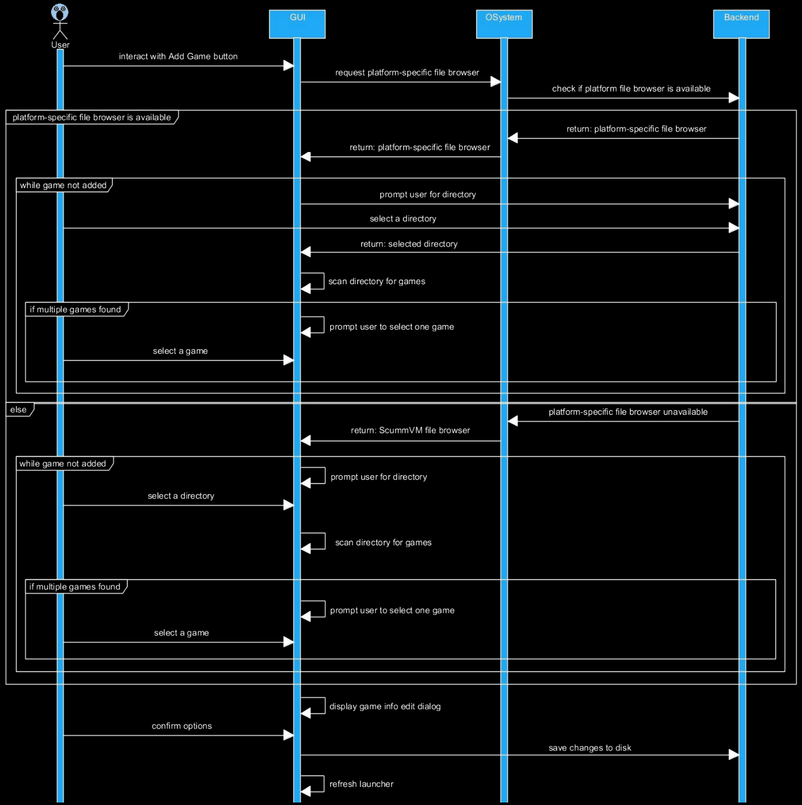
Use case 1: Playing a game - Game loop



Use case 1: Playing a game - Quitting the game

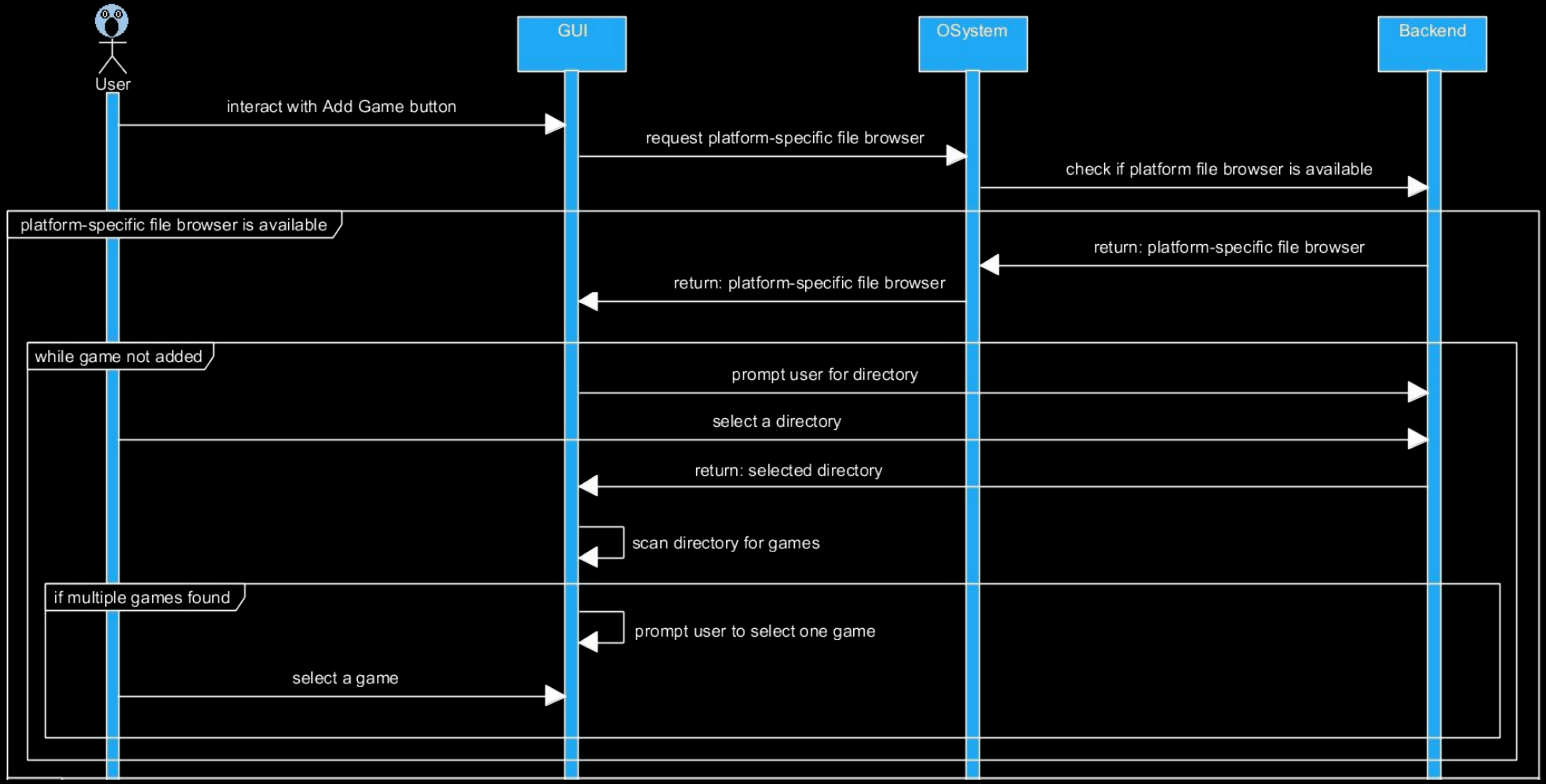


Use case 2: Adding a game

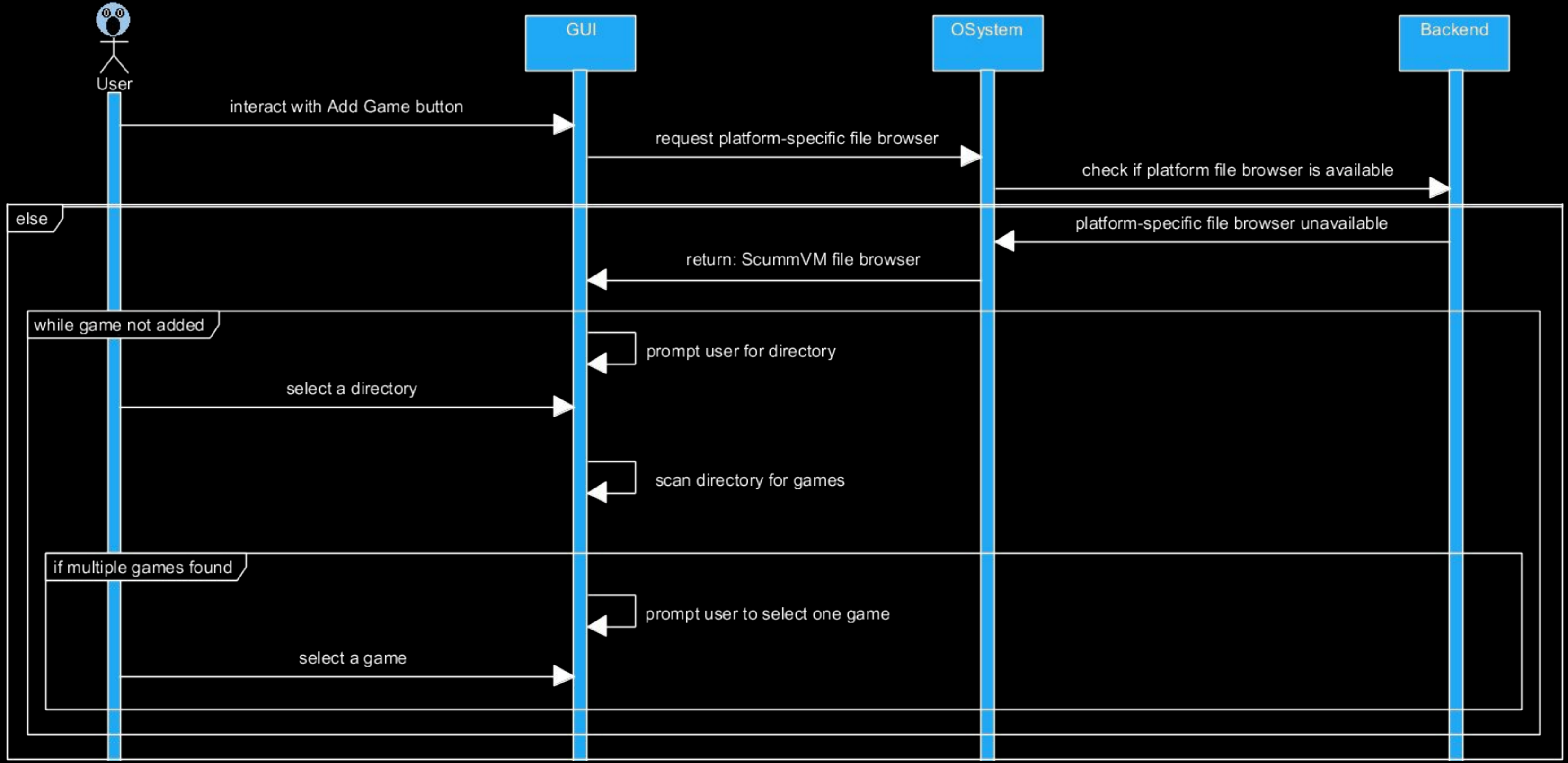

User
GUI
OSystem
Backend
interact with Add Game button
request platform-specific file browser
check if platform file browser is available
platform-specific file browser is available
return: platform-specific file browser
return: platform-specific file browser
while game not added
prompt user for directory
select a directory
return: selected directory
scan directory for games
if multiple games found
prompt user to select one game
select a game
else
platform-specific file browser unavailable
return: ScummVM file browser
while game not added
prompt user for directory
select a directory
scan directory for games
if multiple games found
prompt user to select one game
select a game
display game info edit dialog
confirm options
save changes to disk
refresh launcher

- Two cases
  - The platform-specific file browser is used
  - The ScummVM file browser is used

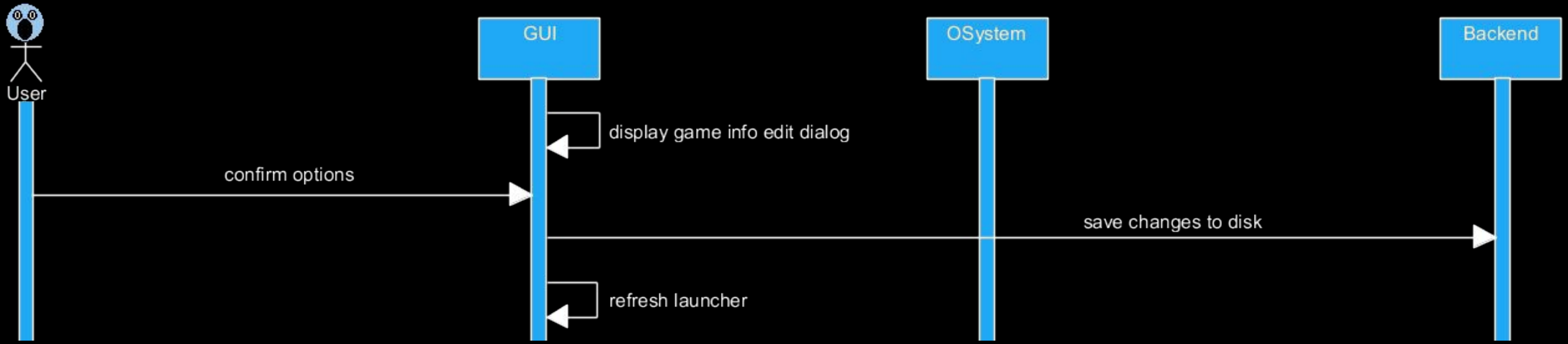
Use case 2: Adding a game - Platform-specific file browser


User
GUI
OSystem
Backend
interact with Add Game button
request platform-specific file browser
check if platform file browser is available
platform-specific file browser is available
return: platform-specific file browser
return: platform-specific file browser
while game not added
prompt user for directory
select a directory
return: selected directory
scan directory for games
if multiple games found
prompt user to select one game
select a game

Use case 2: Adding a game - ScummVM built-in file browser


User
GUI
OSystem
Backend
interact with Add Game button
request platform-specific file browser
check if platform file browser is available
else
platform-specific file browser unavailable
return: ScummVM file browser
while game not added
prompt user for directory
select a directory
scan directory for games
if multiple games found
prompt user to select one game
select a game

Use case 2: Adding a game - Wrapping up



Lessons Learned

- Dependency graphs along with source code enable a fast and effective method for understanding software
  - Dependency graph gives a solid high-level foundation
  - Source code gives a complete low-level understanding
  - Dependency graph can be used to determine where a deep analysis of the source code is needed
- Performing research independently allowed maximum generation of ideas, while minimizing confusion
- Working on sections independently while maintaining communication with the group was a good way of ensuring that the architecture remained cohesive

Conclusion

Through careful examination of the project’s Github repository and analysis of its files with the Understand tool, we developed a stronger conceptual architecture as well as a concrete architecture for ScummVM.

The concrete architecture we derived from these files is composed of four layers which work together along with shared utility modules to form ScummVM. Our research outlines discrepancies and performs a reflexion analysis to elucidate the nature of these differences. The concrete architecture is further illuminated by investigation of the game engines subsystem and its associated reflexion analysis, as well as illustration of two use cases.

The report provides a systematic and comprehensive analysis of ScummVM’s concrete architecture, with detailed discussion of any inconsistencies discovered and careful analysis of the key components required for its functionality.