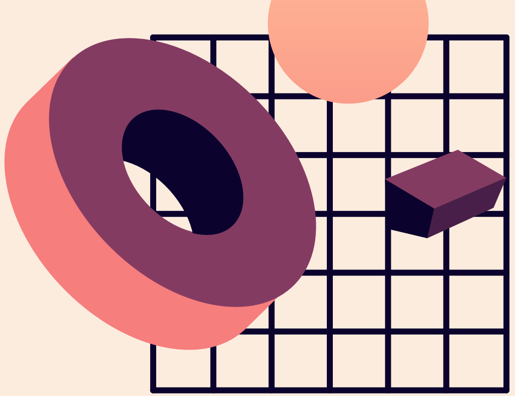


YouTube Video Link

https://youtu.be/N3BUIxDxSQE

Assignment #2

Group #25 - 2024/11/18 - CISC 322/326

Group Lead: Elill Mathivannan

Presenters: Henry Xiu, Amaan Javed

Members: Momin Alvi, Elias Frigui, Ahmad Tahir

Agenda

- Derivation Process
- Conceptual Architecture I & II
- Concrete Architecture I & II (Overall)
- Concrete Architecture (Game Engine Layer)
- Reflexion Analysis
- Use Cases
- Lessons Learned
- Conclusion

Derivation Process

1) Analyze the dependency file
    a) SciTools Understand
        i) Dependency Graph
2) Organize system into three layers
    a) Break into subsystems
    b) Repeat for each component

Conceptual Architecture I

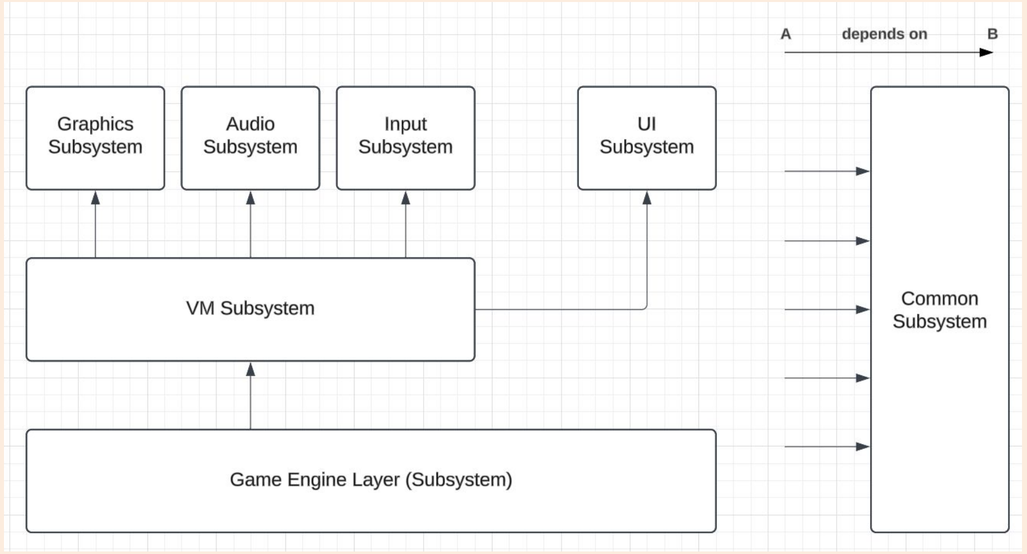


Figure 1. ScummVM Conceptual Architecture and Subsystem Interactions

Conceptual Architecture II

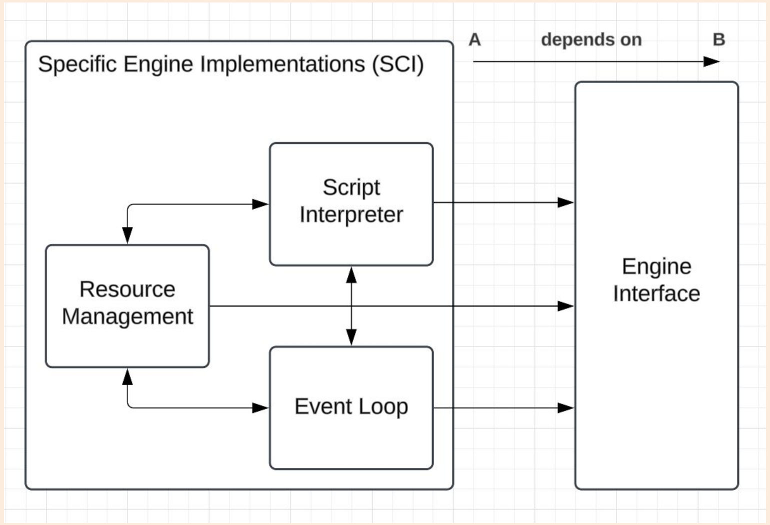


Figure 2. Detailed Architecture of the Game Engine Subsystem in ScummVM

Concrete Architecture I

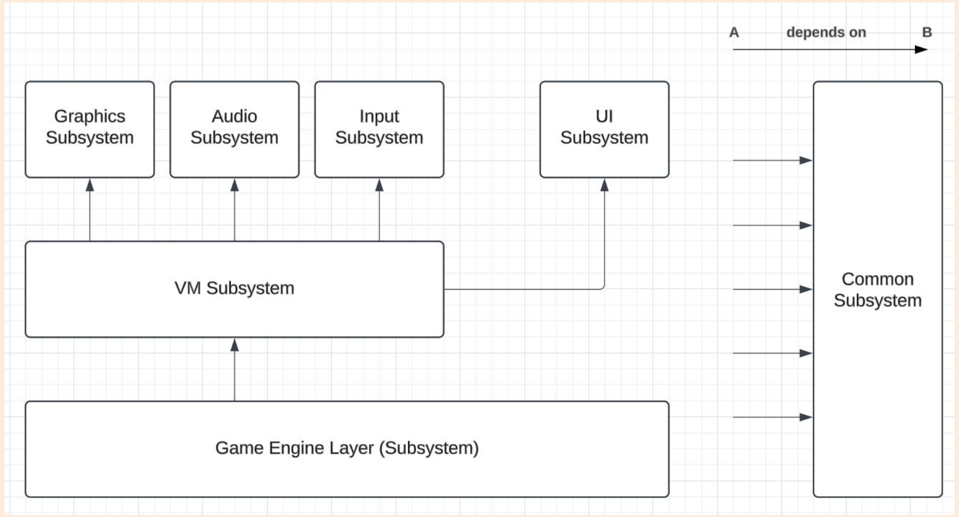


Figure 3. ScummVM Conceptual Architecture and Subsystem Interactions

Concrete Architecture II

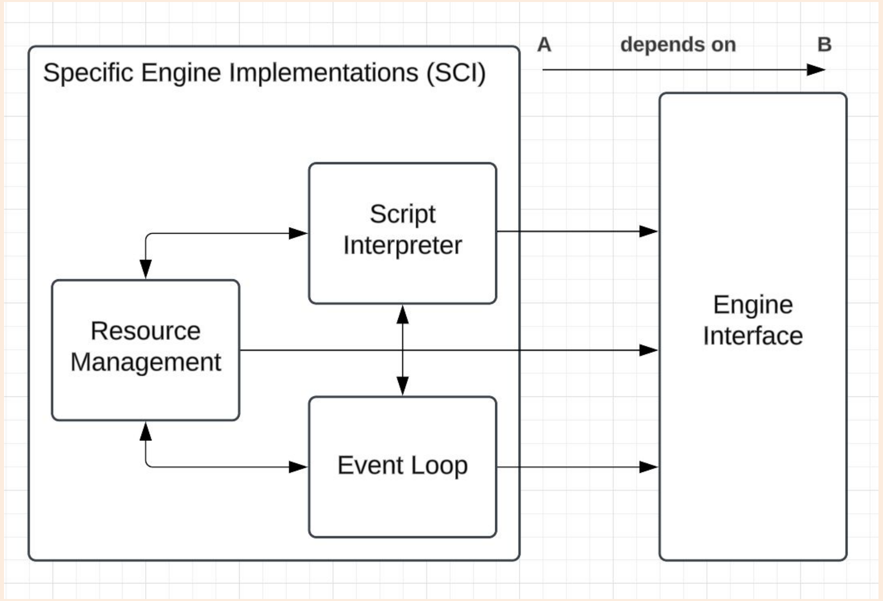


Figure 4. Detailed Architecture of the Game Engine Subsystem in ScummVM

Concrete Architecture (Game Engine Layer)

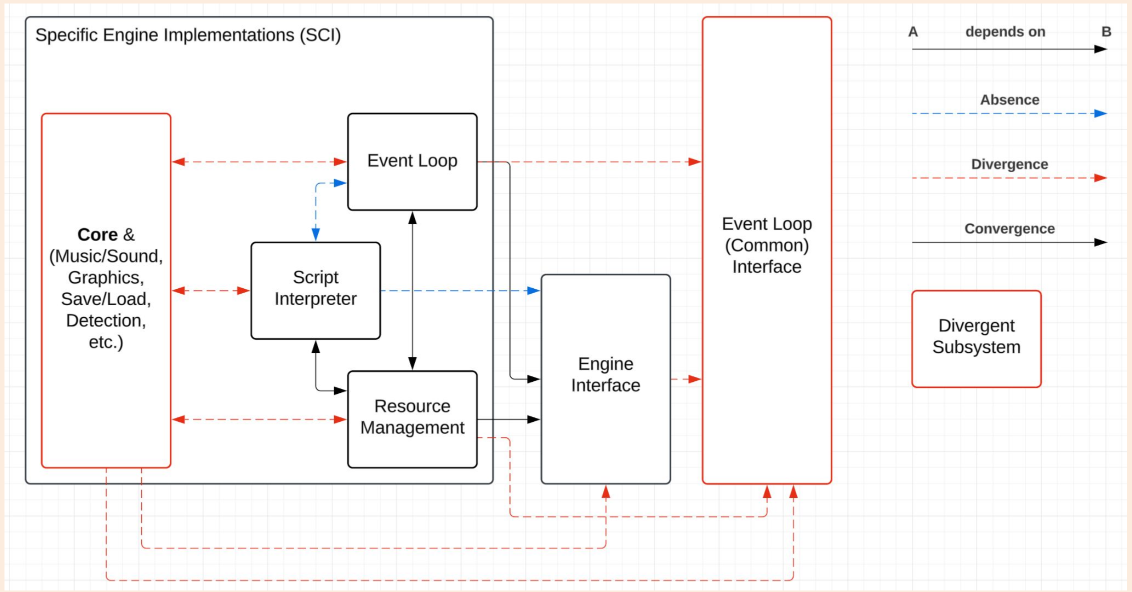

Specific Engine Implementations (SCI)
Core & (Music/Sound, Graphics, Save/Load, Detection, etc.)
Event Loop
Script Interpreter
Resource Management
Engine Interface
Event Loop (Common) Interface
A
depends on
B
Absence
Divergence
Convergence
Divergent Subsystem

Figure 5. ScummVM Game Engine Concrete Architecture and Subsystem Interactions

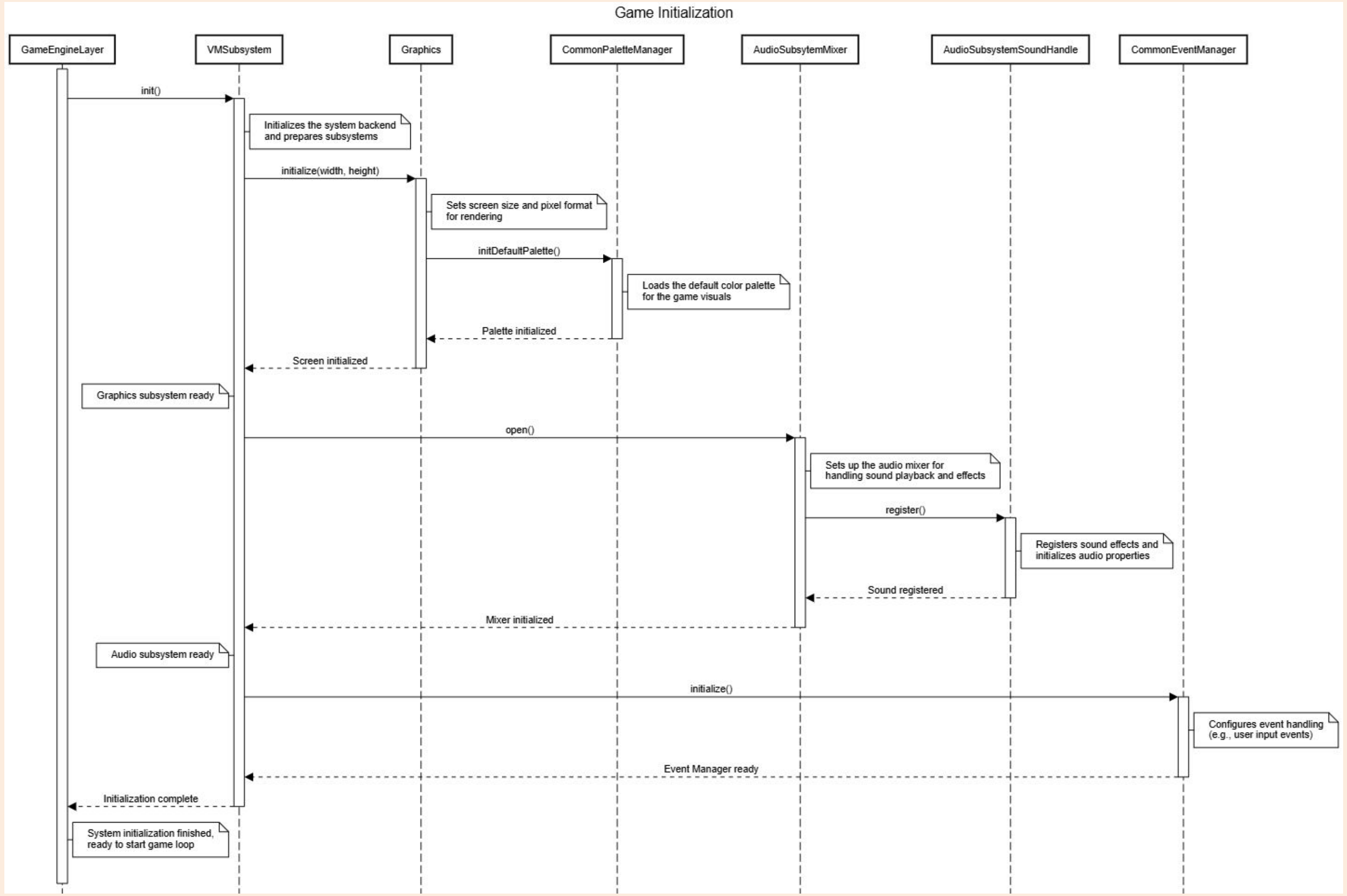
Reflexion Analysis

- **Convergences**
  - Example: Graphics, Audio, Input, Common subsystems retained modularity
- **Divergences**
  - Example: Game Engine Layer’s reliance on the Input Subsystem
- **Absences**
  - Example: Game Engine Layer and Common Subsystem

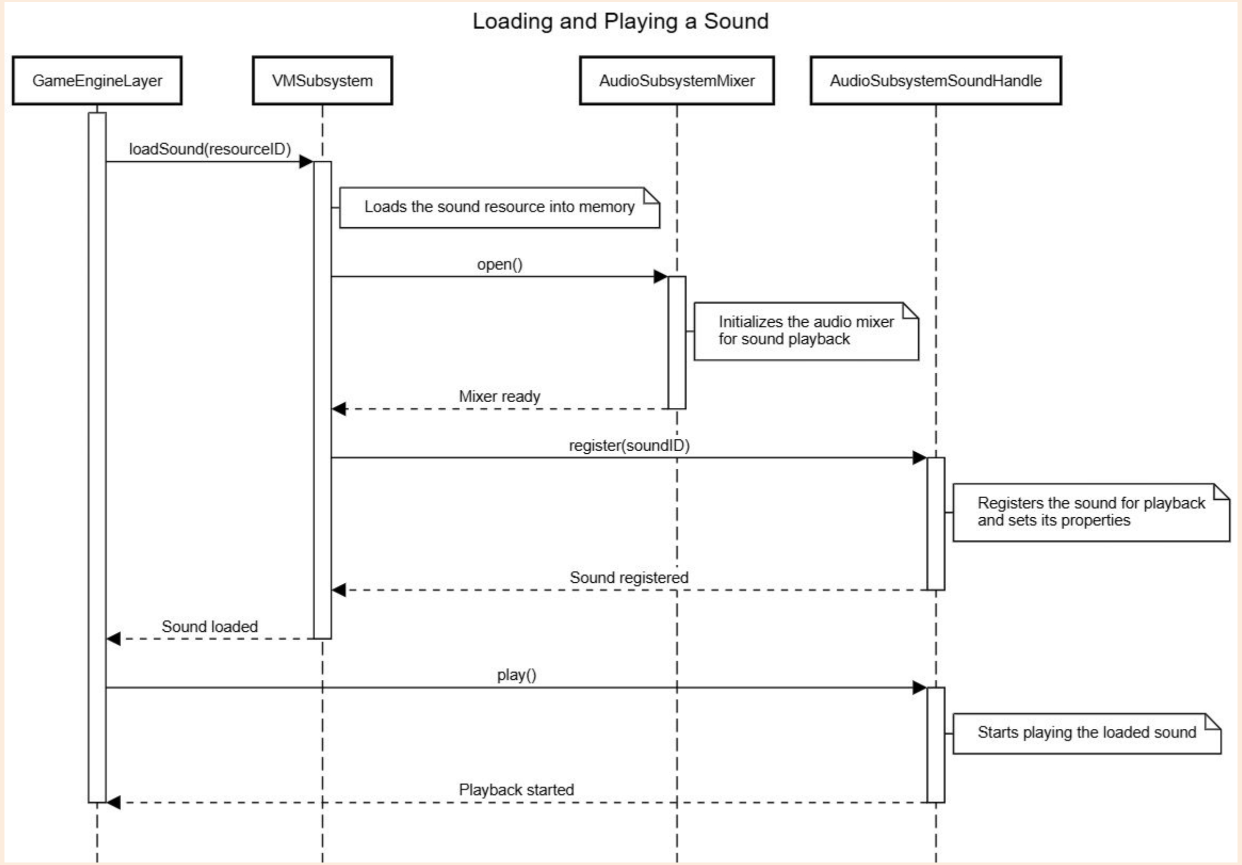
Use Cases

- We will look at two use cases:
  - Game Initialization
  - Loading and Playing Sounds

Game Initialization Use Case


Game Initialization
GameEngineLayer
VMSubsystem
Graphics
CommonPaletteManager
AudioSubsytemMixer
AudioSubsystemSoundHandle
CommonEventManager
init()
Initializes the system backend and prepares subsystems
initialize(width, height)
Sets screen size and pixel format for rendering
initDefaultPalette()
Loads the default color palette for the game visuals
Palette initialized
Screen initialized
Graphics subsystem ready
open()
Sets up the audio mixer for handling sound playback and effects
register()
Registers sound effects and initializes audio properties
Sound registered
Mixer initialized
Audio subsystem ready
initialize()
Configures event handling (e.g., user input events)
Event Manager ready
Initialization complete
System initialization finished, ready to start game loop

Loading and Playing Sounds


Loading and Playing a Sound
GameEngineLayer
VMSubsystem
AudioSubsystemMixer
AudioSubsystemSoundHandle
loadSound(resourceID)
Loads the sound resource into memory
open()
Initializes the audio mixer for sound playback
Mixer ready
register(soundID)
Registers the sound for playback and sets its properties
Sound registered
Sound loaded
play()
Starts playing the loaded sound
Playback started

Lessons Learned

- SciTools Understand
  - Learning Curve
- Clear and comprehensive documentation
- More effective role assignment
- Documenting trade-offs

Conclusion

- System maintains modularity
- System adheres to design principles
- Notable divergences, convergences, absences
- To Improve
  - Regular architectural reviews and iterative updates
  - Optimize subsystem interactions
  - Addressing missing dependencies

Thank you