https://drive.google.com/file/d/1lB1F-s53sEy7AYgfC4dbu62wrzjpFZP5/view?usp=sharing

CONCEPTUAL ARCHITECTURE

For ScummVM and SCI Engine

Trevor White (**Group Leader**, Conclusions, SCI Components, Use Cases)

Sophia Pagazani (**Presenter**, SCI Components, Architectural Style)

Aniss Hamouda (**Presenter**, ScummVM Components, Use Cases)

Claire Whelan (Introduction, External Interface)

Nicole Hernandez (ScummVM Components, System Evolution)

Nasreen Mir (Derivation Process, External Interface)

TABLE OF CONTENTS

INTRODUCTION

ScummVM:

- Script Creation Utility for Maniac Mansion Virtual Machine
- Allows users to run classic adventure games on foreign systems
- Focused on addressing portability by expanding support for more engines
- Game engines are rewritten to run in a platform non-specific environment with ScummVM being the intermediary

SCI:

- Script Code Interpreter/Sierra's Creative Interpreter
- Game engine ScummVM supports
- Mainly for adventure and text-based games

This Presentation:

- An architectural overview view of both systems and their interactions
  - And how we got there!
- Use cases for the systems
- Conclusions and what we got out of this
- And more!

DERIVATION PROCESS

Process:

- Consulted various documentations for each system
- Split the team into two groups: one for ScummVM and one for SCI
- We settled on Layered Style for ScummVM and Layered & Interpreter Style for SCI
  - More detail in Architectural Overview!

Alternatives:

- We considered that ScummVM might also use the interpreter style, but we decided that it makes more sense for just the SCI engine

ARCHITECTURAL OVERVIEW

SCUMMVM

User Interface

Bridge between User and ScummVM

Game Database

Stores game files and save files

Common API

APIs that support many other components

Output API

APIs that process/pass graphics and audio

Backends

Allow ScummVM to run on many platforms

SCI

Graphics Subsystem

Handles all visual displays for a game in SCI

Sound Subsystem

Manages all music and sound for a game in SCI

Window Manager

Invisible but holds all other windows and ports

Device Drivers

Takes input and produces events, then given to event handler

Heap and Hunk

Used for memory allocation within the engine

SCI

Event Manager

Receives events and activates the related subsystems

Parser

Takes all user-entered text and compares it to a vocab list

Animation Subsystem

Has a mover and a cycler that executes animations

Game Code

All scripts and objects that make up a game in SCI

Interpreter

Uses all subsystems to allow game code to be executed

ARCHITECTURAL STYLE

ScummVM has a Layered architectural style:

- Backend Tier: backends/ports, database for game files
  - Lowest level, communicates with the middleware tier
- Middleware Tier: Output API, Common API
  - Middle level, communicates with both backend tier and frontend tier
  - Intermediary for the two other tiers
- Frontend Tier: User Interface, Game Engines (like **SCI**)
  - Highest level, communicates with the middleware tier
  - Focused on rendering and extending middleware APIs

ARCHITECTURAL STYLE

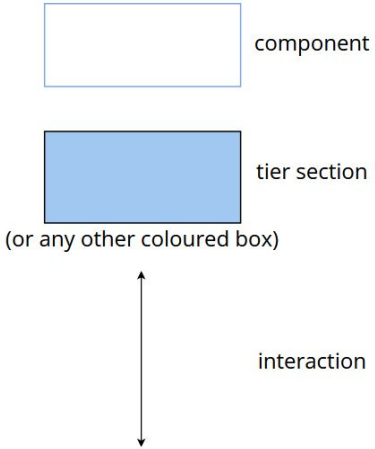
SCI while be in one of ScummVM's tier, has its own distinct architectural styles. The first being also Layered style:

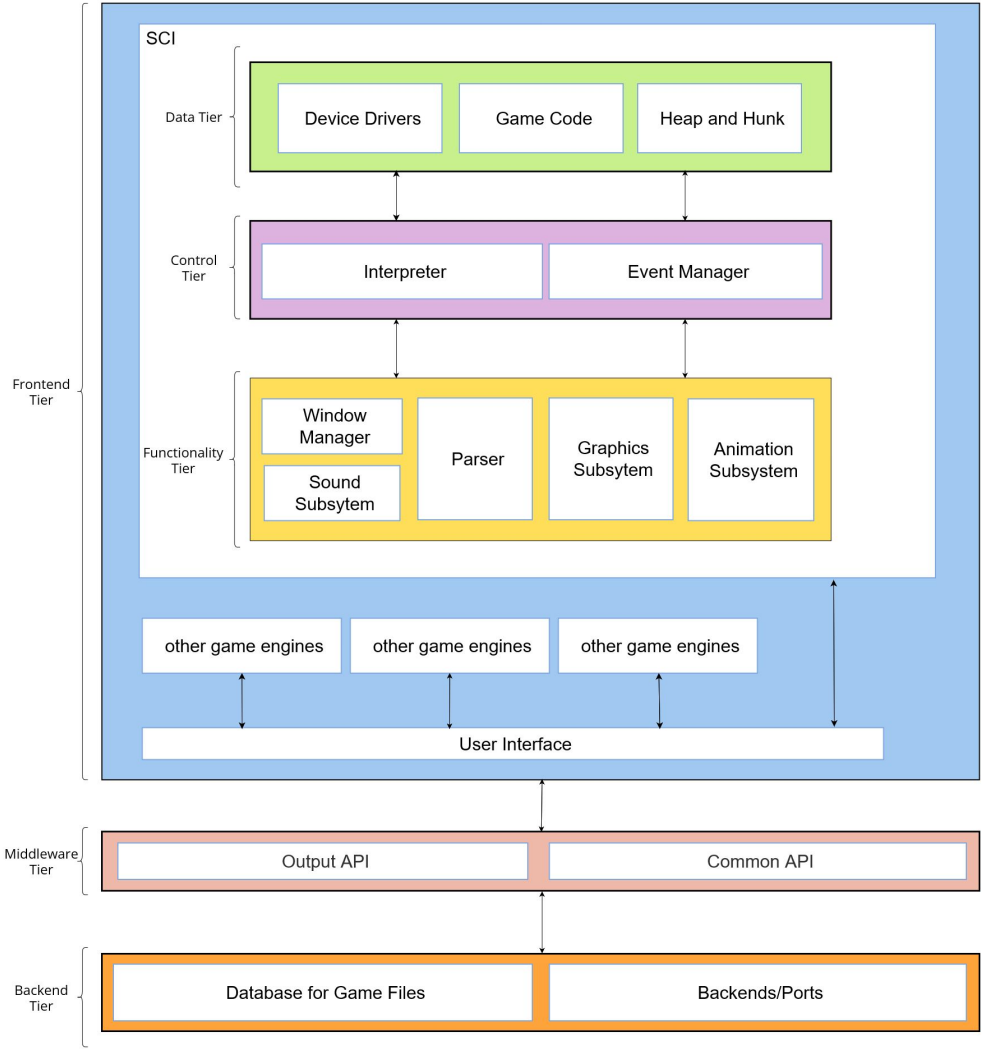
- Functionality Tier: Window Manager, Parser, Graphics Subsystem, Sound Subsystem, Animation subsystem
  - Lowest level, communicates with control tier
  - Concerned with all visual parts of a game and actually running it
- Control Tier: Interpreter, Event Manager
  - Middle level, communicates with both functionality tier and data tier
  - Most important, how everything is allowed to run
- Data Tier: Device Drivers, Game Code, Heap and Hunk
  - Highest level, communicates with control code
  - All resources required for game to run

ARCHITECTURAL STYLE

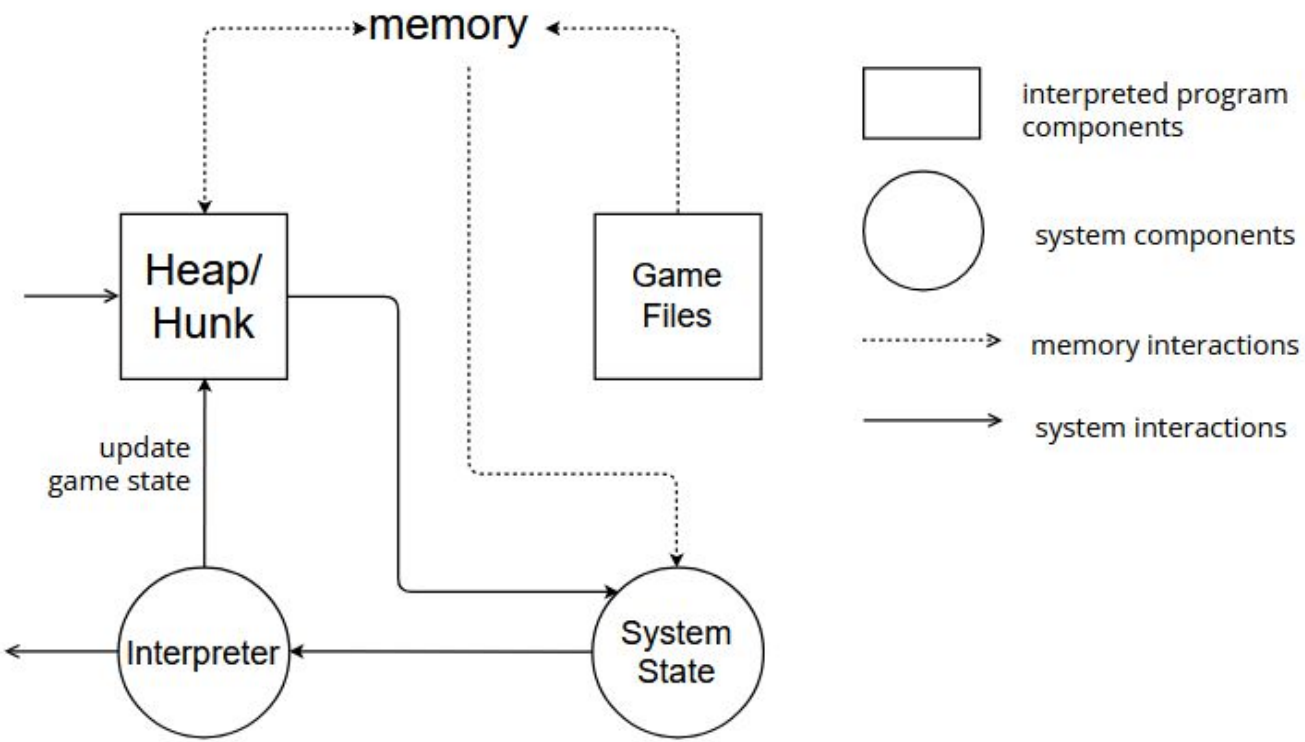
SCI also has an Interpreter style architecture:

- Has two Domain Specific Languages (DSLs) to extend the program
- Four Components to make up the style:
  - "The execution engine" = the main game loop
  - "Current state of the execution engine" = communicated to the interpreter by the timer component
  - "The program being interpreted" = any of the games created for the engine (i.e., King's Quest IV)
  - "Current state of the program being interpreted" = the current game state





System Architecture Box and Line Diagram



Interpreter Flow Diagram

EXTERNAL INTERFACE

ScummVM acts to transmit information between the engine and the system it is running on
It transmits information using APIs related to:

- Event handling
- Audio and Visual Output
- The file system

The **engine** also receives information from the GUI, namely:

- User Input
- Display Updates
- State Changes

Internally, the **engine** transmits game files relating to:

- Graphics
- Sounds
- User Interaction.

ScummVM also has its own files like:

- Configuration Files
- Plugins
- Translation Files
- Save Directory

USE CASES

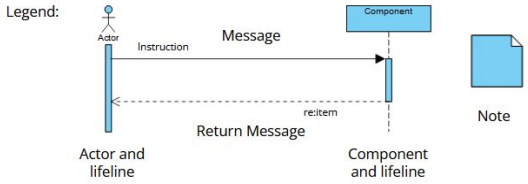
LAUNCHING A GAME

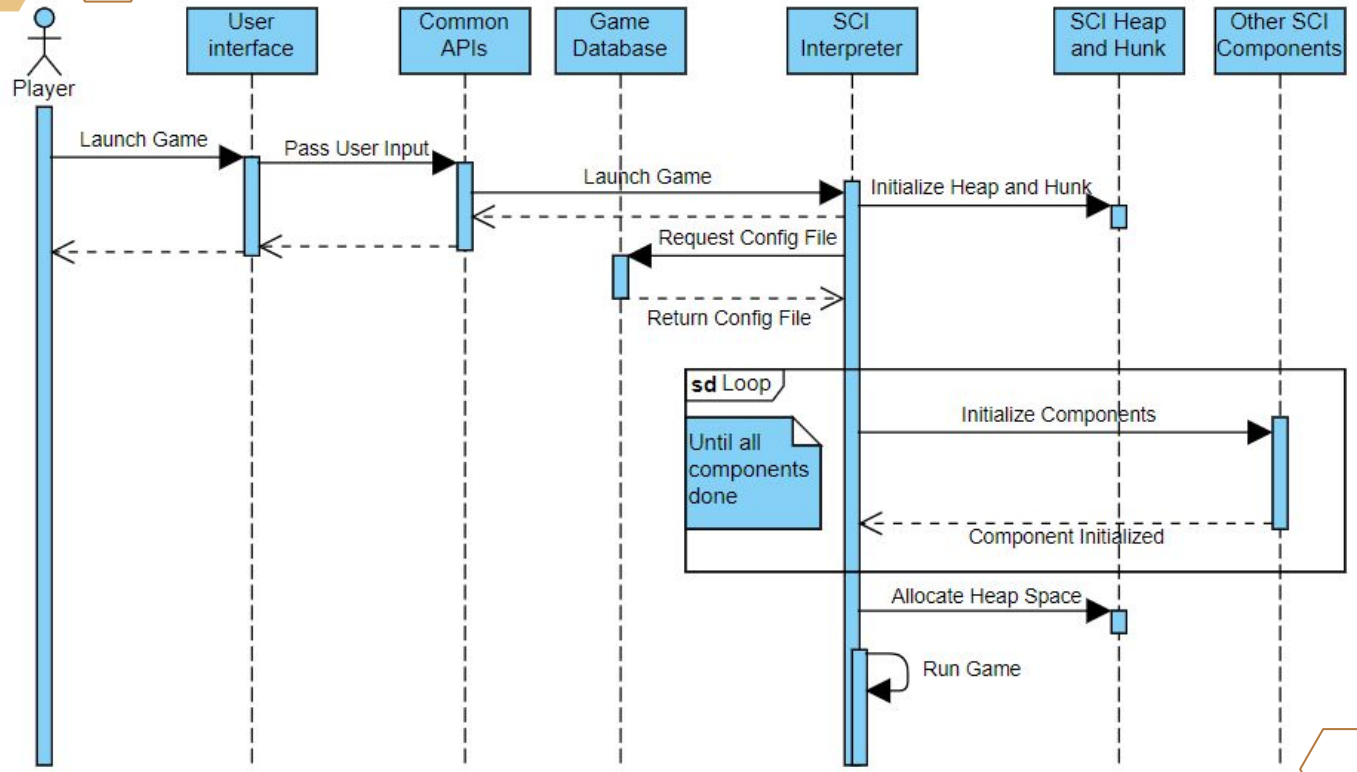
Once the player launches the game, ScummVM passes this to the SCI Interpreter.

The Interpreter will:

- Initialize all components
- Parse Config Files and load drivers
- Allocate space on heap
- Run Game

The interpreter is always active and acts as the main controller for the engine





Launching a Game

USING A TEXT BASED COMMAND

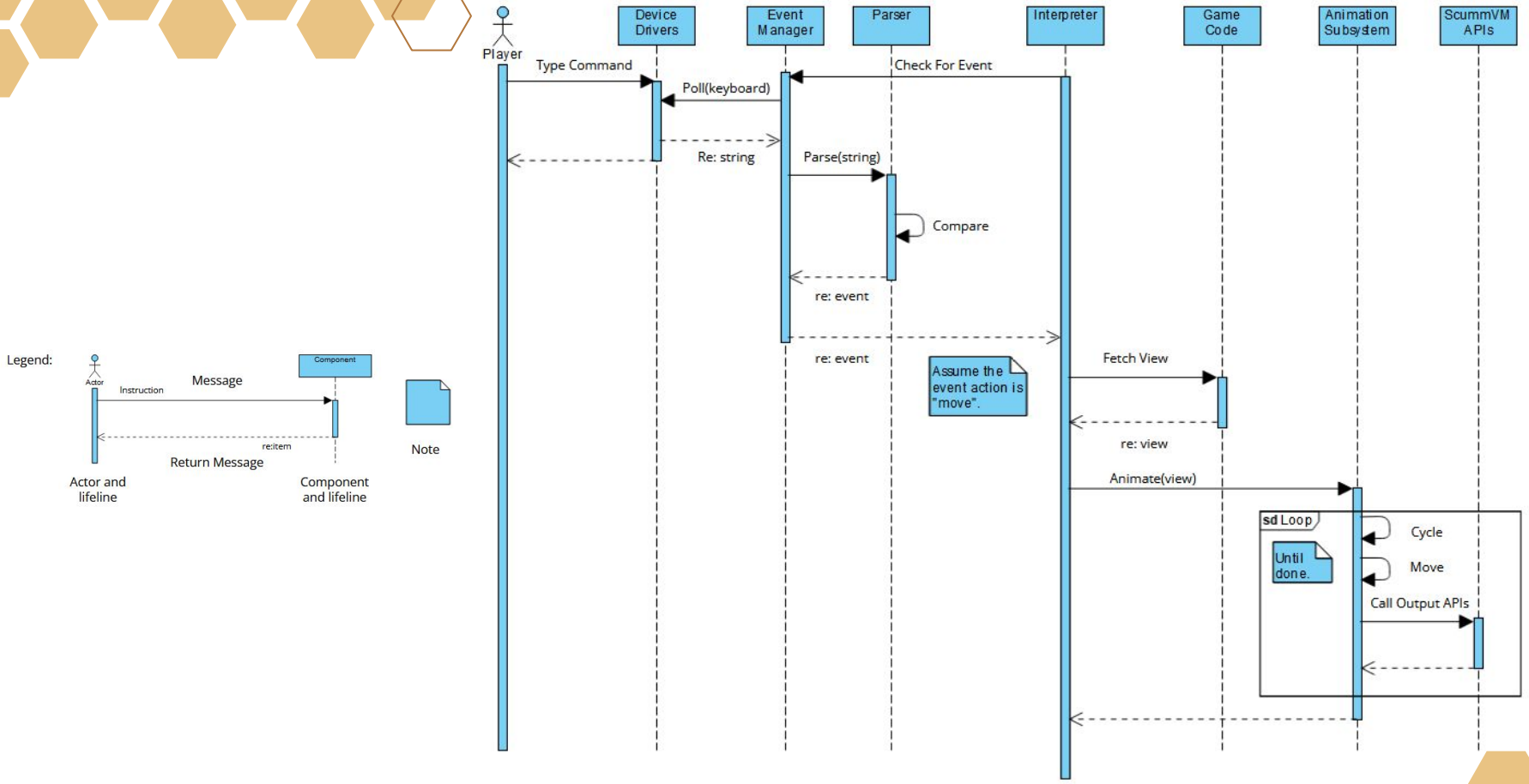
When the user inputs a string:

- String passed to event manager
- String passed to parser
- Parser generates event, returns it
- Interpreter requests event

Interpreter will then:

- Request view from game code
- Dispatch to animation subsystem.

Animation subsystem will loop cycler, mover and Output APIs



Text Based Command

EVOLUTION OF THE SYSTEM

ScummVM is an open source project, so anyone can contribute to it as long as they align with the coding conventions. All the source code can be found in the Github repository. In order to do so you must:

- Code in C++ (exceptions being dev tools, Apple backends, android backends)
- Fork the repository
- Go through the Pull request system
- Have the **ScummVM** team review your changes for two weeks

If the **ScummVM** team has no objections, your additions will be merged with the existing code, and you must document the changes you made.

LESSONS LEARNED

Learning how **Open Source Projects** are built and maintained

Learning about **APIs** and how they work

Understanding architectural styles, especially **Interpreter style**

Our biggest **mistake** was treating **ScummVM** and the **SCI engine** as 2 separate subsystems, instead of one unified system

CONCLUSION

ScummVM allows game engines to run on non natives system using a **layered** architecture, comprised of:

- UI elements and game files, allowing the user to launch and save games
- Many APIs, that facilitate communication between the engines and the system running it

The **SCI engine** is a **layered** system. It also has an **interpreter** style architecture, having a component that:

- Directs the other components
- Tracks the state of execution

These findings are all **conceptual**, as they are derived from documentation. The next step is to look at the source code to determine a **concrete architecture.**

slidesgo