https://drive.google.com/file/d/1OvyVWKhZQuWoTSg6qtGpZEr9Pc0lDDvm/view?usp=sharing

CONCRETE ARCHITECTURE

For ScummVM and SCI Engine

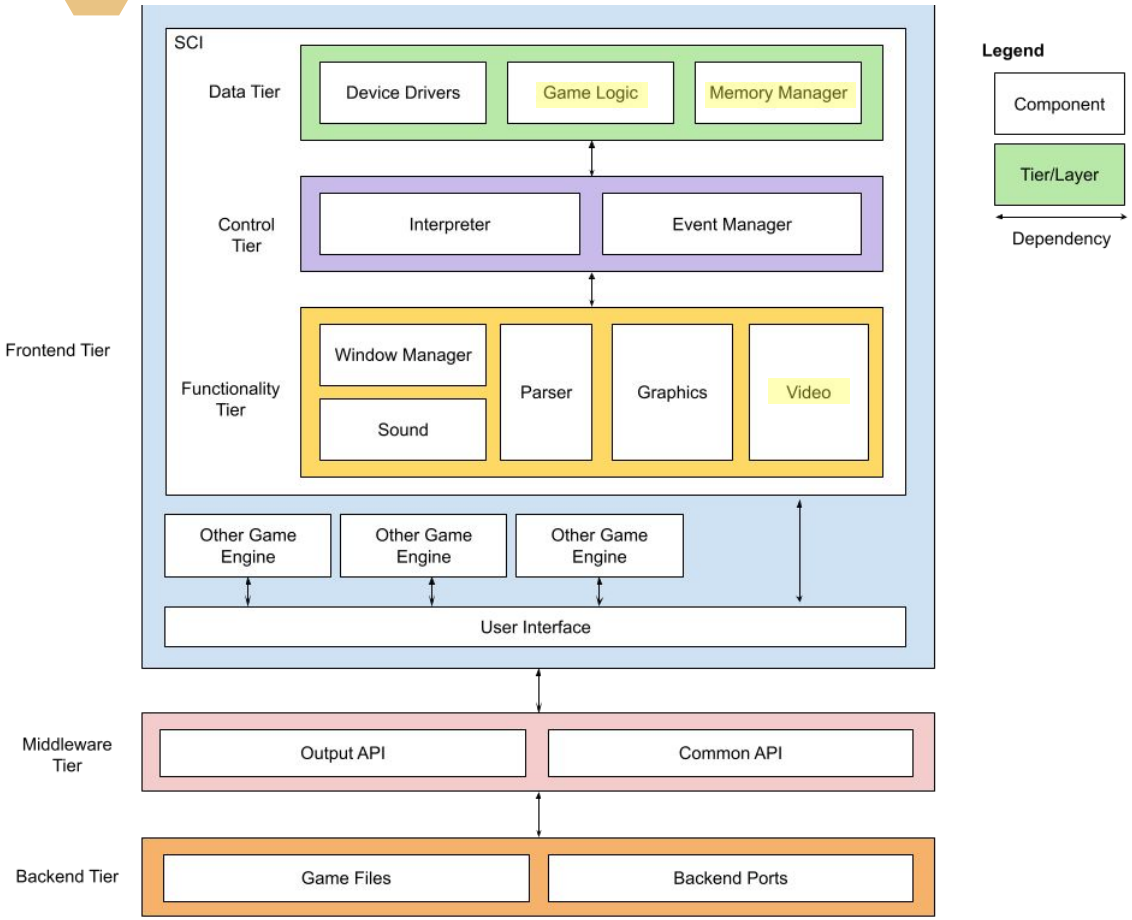
Trevor White (**Group Leader**, Concrete Architecture, Use Cases)

Sophia Pagazani (**Presenter**, SCI Reflexion Analysis)

Aniss Hamouda (**Presenter**, Use Cases, Conclusions)

Claire Whelan (Abstract, High-Level Reflexion Analysis)
Nicole Hernandez (Derivation Process, Lessons Learned)
Nasreen Mir (Introduction, Review of Conceptual Architecture)

TABLE OF CONTENTS



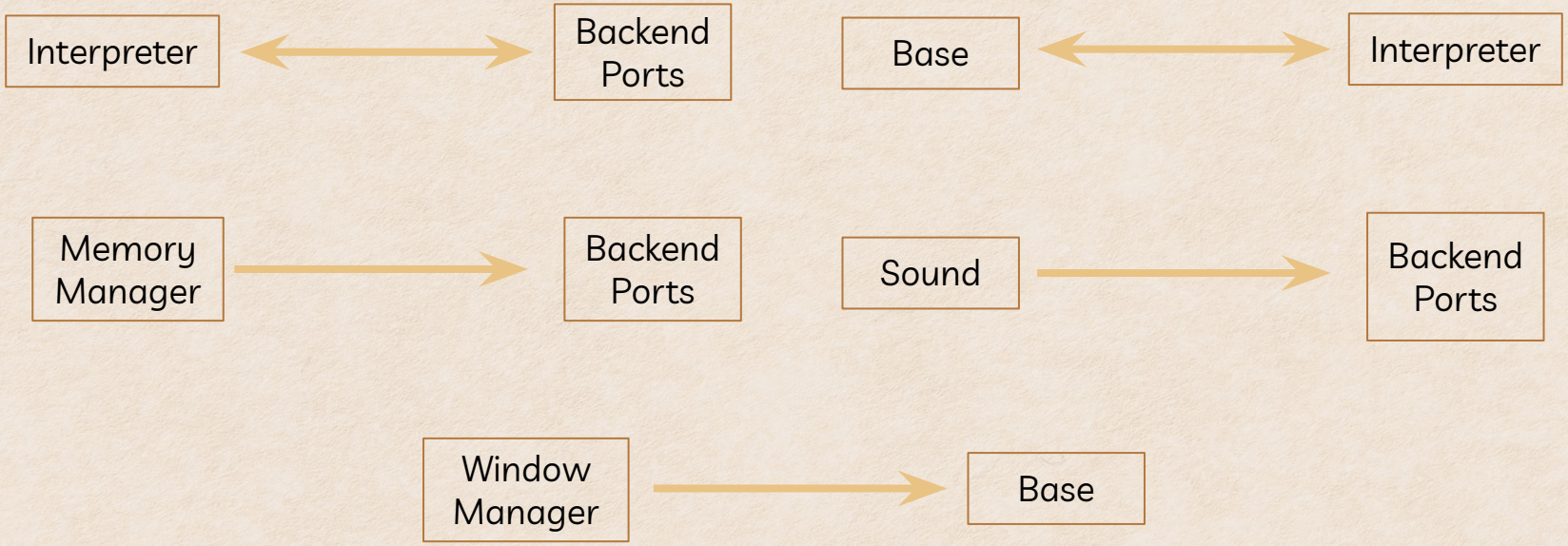
Review of Conceptual Architecture

DERIVATION PROCESS

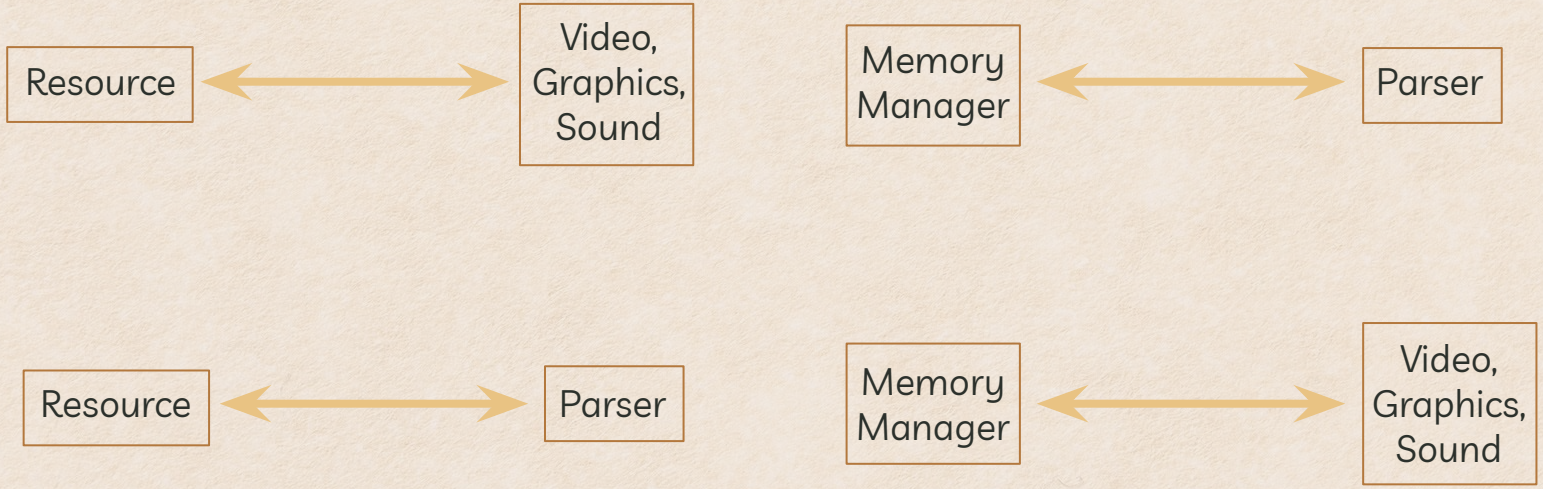
1. Each member loaded files into Understand
2. With our conceptual architecture as a base, mapped code to components
3. Discussed placements as a group, renamed/added components as needed
4. Conducted a reflexion analysis and identified divergences
5. Derived a concrete architecture based on our findings

REFLEXION ANALYSIS

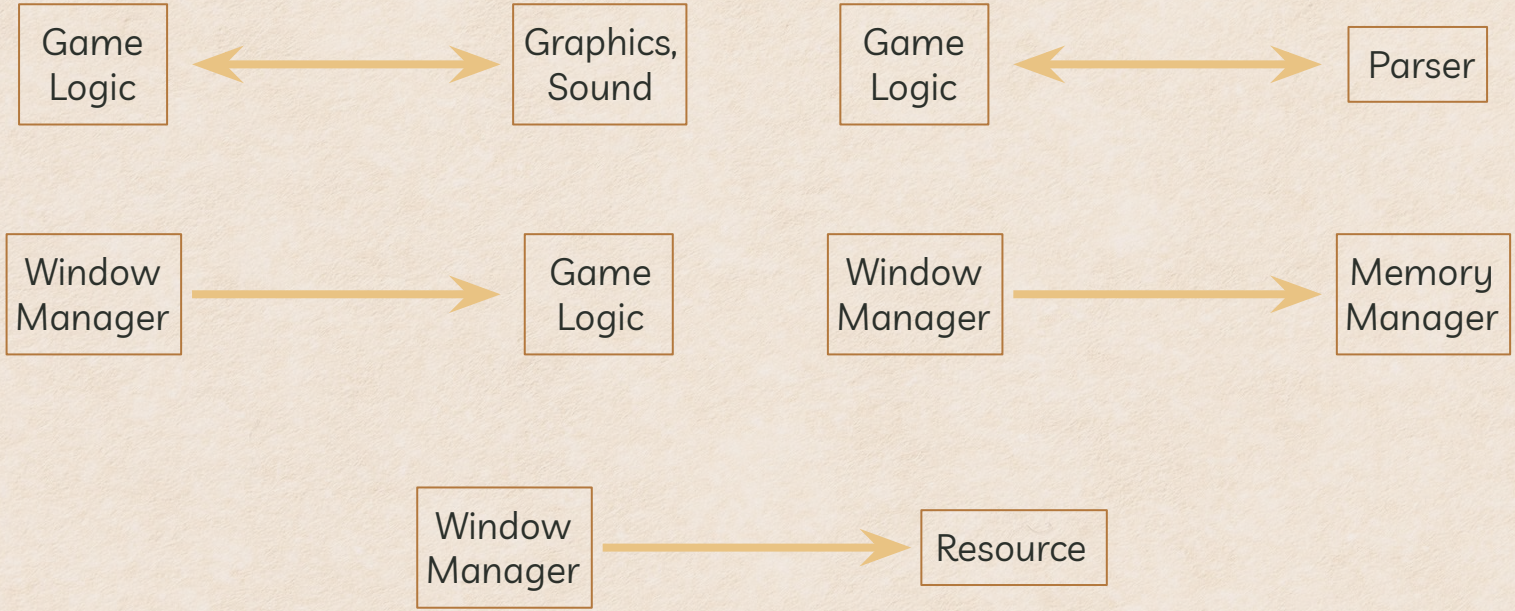
TOP-LEVEL


Interpreter
Backend Ports
Base
Interpreter
Memory Manager
Backend Ports
Sound
Backend Ports
Window Manager
Base

WITHIN SCI


Resource
Video, Graphics, Sound
Memory Manager
Parser
Resource
Parser
Memory Manager
Video, Graphics, Sound

WITHIN SCI


Game Logic
Graphics, Sound
Game Logic
Parser
Window Manager
Game Logic
Window Manager
Memory Manager
Window Manager
Resource

CONCRETE ARCHITECTURE

DERIVED ARCHITECTURE

Problems:

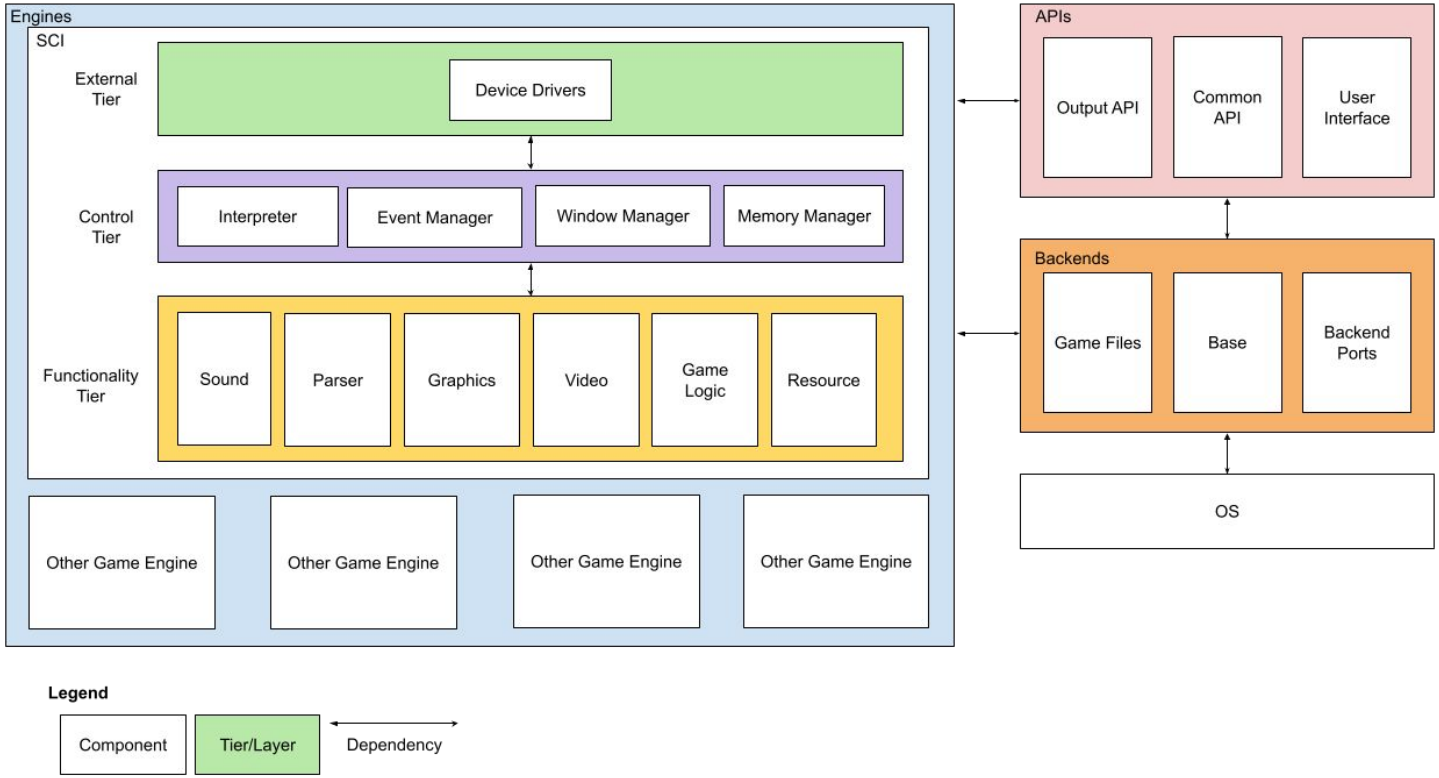
- Divergent dependencies violated original layering
- Component gaps became evident
- Misappropriated components also discovered

Solutions:

- Added Base and Resource
- Renamed:
  - Animation → Video
  - Game Code → Game Logic
  - Heap/Hunk → Memory Manager
- Redefined tiers
- Restructured layers

Observations:

- Maintains both architectural styles
- Pub-Sub for SCI as well
- Maintains our initial findings and conclusions



Concrete Architecture

USE CASES

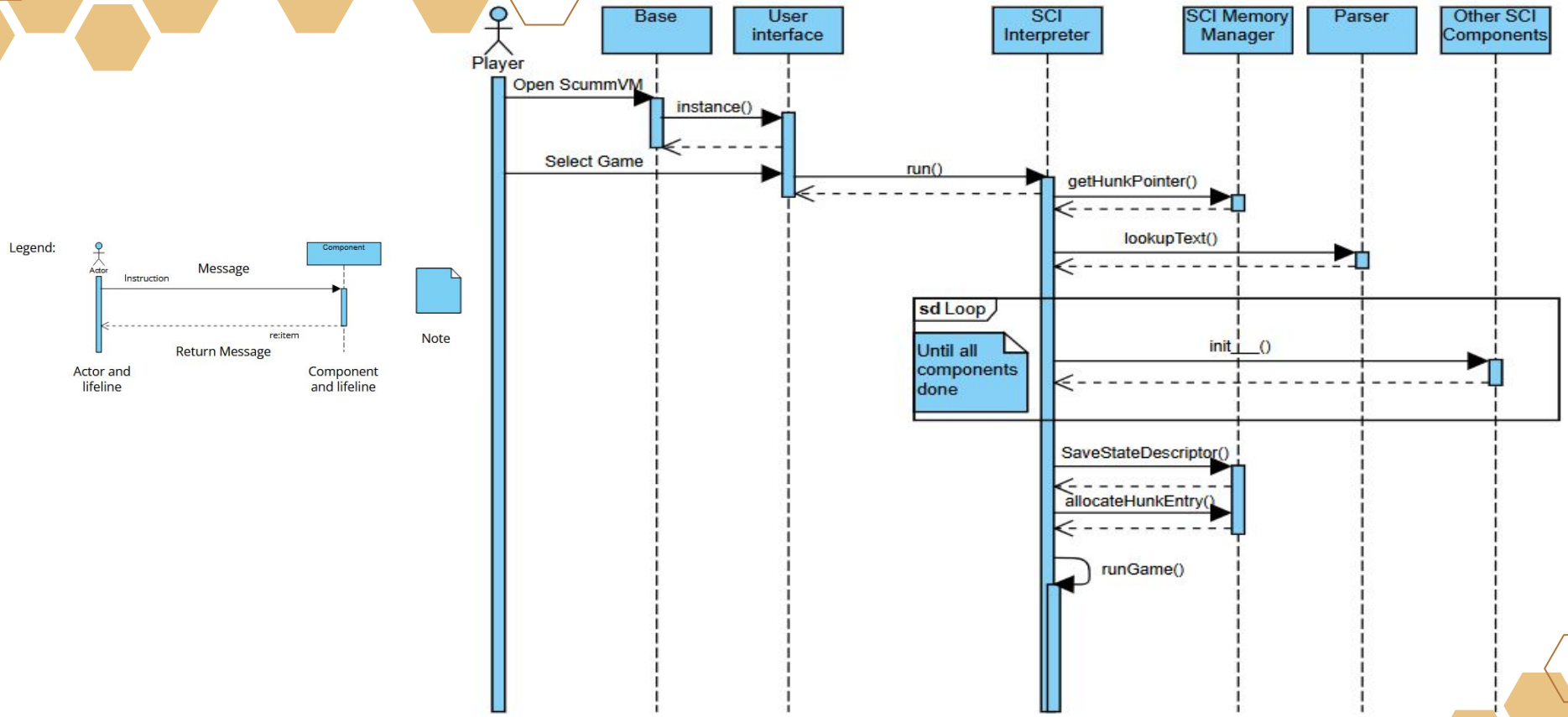
LAUNCHING A GAME

When the player opens ScummVM, the Base component creates an instance of the User Interface, where the user can select a game. Once the player launches the game, ScummVM passes this to the SCI Interpreter.

The Interpreter will:

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

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



Launching a Game

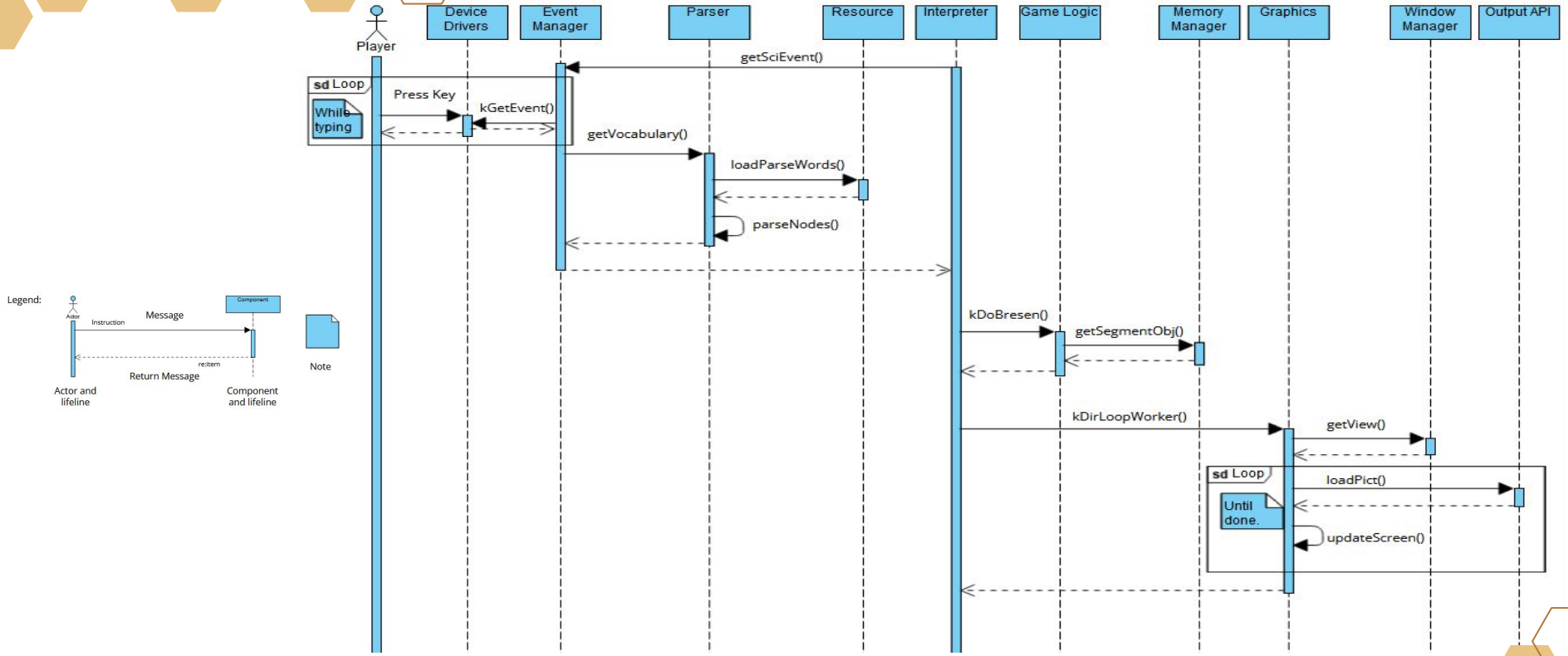
USING A TEXT BASED COMMAND

When the user inputs a string:

- Interpreter queries Event Manager
- Event Manager passes string to Parser
- Parser generates event, returns it
- Event manager passes to Interpreter

Interpreter will then:

- Call Game Logic to determine path and game object
- Call Graphics to render movement on screen
  - Graphics will get view space from window manager
  - Graphics will loop, calling Output API



Text Based Command

CONCLUSION

Using Understand, we were able to develop a **concrete** architecture for the system. This was fairly similar to the conceptual architecture

Divergences:

- **New** Resource and Base **components**
- Tier changes
  - Frontend -> Engine
  - Middleware -> API
- User interface moved to API tier
- API and Backend work side by side instead of in hierarchy

By analyzing source code, we got a better understanding of the system.
We can use this to suggest **improvements** to the system.

LESSONS LEARNED

Teamwork:
We did not split up the system and all worked together

Documentation:
It was a challenge to understand code with minimal documentation

Proper Tools:
Using Understand helped find the dependencies