PROPOSAL FOR ENHANCEMENT OF SCUMMVM

CISC 322/326 A3 Report

2 December 2024

Team MAWLOK
Lance Lei (20lh10@queensu.ca)
Michael Marchello (21mgm13@queensu.ca)
Owen Meima (21owm1@queensu.ca)
Kevin Panchalingam (15kp20@queensu.ca)
Andrew Zhang (21az75@queensu.ca)
Zhuweinuo Zheng (21zz28@queensu.ca)

TABLE OF CONTENTS

ABSTRACT

This report will outline the design and thought process behind a multiplayer framework enhancement for the ScummVM software. Using our knowledge, developed through creating the conceptual and concrete architecture for the software, we propose an upgrade to the OSystem API, enabling multiplayer functionality for future games and engines. Our derivation process involved brainstorming potential features, evaluating alternatives, and analyzing the conceptual and concrete architectures of ScummVM. Through this process we have a new networking component, responsible for connecting to a server, into the backend layer. Two design alternatives for our new feature were explored: A host-player system and a remote-host system. Each alternative was assessed for usability, performance, maintainability, and other non-functional requirements, emphasizing their impacts on stakeholders such as players, developers, and server owners. The framework's design and sequence diagrams demonstrate the interaction between components, while a risk analysis highlights potential challenges, including server bottlenecks and connectivity issues. Throughout our process, we gained valuable insights into software architecture, network communication, and integrating new features into existing systems.

INTRODUCTION

In our A2 deliverable of this project, we took a deeper dive into ScummVM by deriving a concrete architecture based on our conceptual architecture. We proceeded to investigate and understand convergences, divergences and absences between our concrete and conceptual architectures to solidify our understanding of how data and control flowed through the ScummVM system. While ScummVM contains many features that are all well designed and utilized, we consider ways to improve the software in this next phase, the A3 deliverable. We propose a multiplayer backend framework that would benefit stakeholders, further connecting them through gameplay within the ScummVM community. We will walk you through our derivation process of this feature, our consideration of alternatives, the impact of identified non-functional requirements on the alternatives and stakeholders, and the valuable lessons we learned.

OVERVIEW OF MULTIPLAYER FRAMEWORK

The core concept of our architectural enhancement is an upgrade of the OSystem API, providing multiplayer support to future ScummVM engines and games. The current role of the OSystem API is to provide the upper software layers (engines, virtual machines, etc) with a variety of input and output tools which may be used in the process of executing game files. The OSystem API is additionally a superclass of all the various backend implementations of ScummVM, which differ by platform. As described in our previous reports, the backend layer of the ScummVM software consists of all code and components which are platform-dependent (i.e. ‘ScummVM for iOS’). By contrast, all software layers situated above the backend are not unique to the running platform; they are *universal* to all various distributions of ScummVM. Thus, the presence of the OSystem API allows game engines to be built without concern of all *exact* low-level processes by which games may (for example) receive keystrokes and print output images. Rather, engine and game developers must only concern themselves with the general logic that ties input and output together across the duration a game is running.

The purpose of our enhancement is to build a multiplayer framework into the backend of ScummVM, so that new engines and games may utilize multi-platform communication without the hassle of having to implement the entire client-server architecture from scratch. The present-day OSystem API allows individual developers to not be concerned with OS-level input, output, and file management implementation (that would be universal to all games/engines). Our upgrade to the API seeks to add multiplayer communications to this list, so that a future ScummVM game/engine developer may simply define the server-side data and features they want in their product, while having hosting, synchronization, and communication functionality already implemented.

Below is our updated conceptual architecture diagram, featuring the three-layer structure discussed in our previous report. This new diagram includes the alterations made during our concrete architecture investigation, as well as an entirely new ‘Networking Utilities’ component (which contains the core functionality of our enhancement). This new component interacts with the I/O components, user interface, and game engines in order to facilitate multi-platform execution of games on ScummVM.

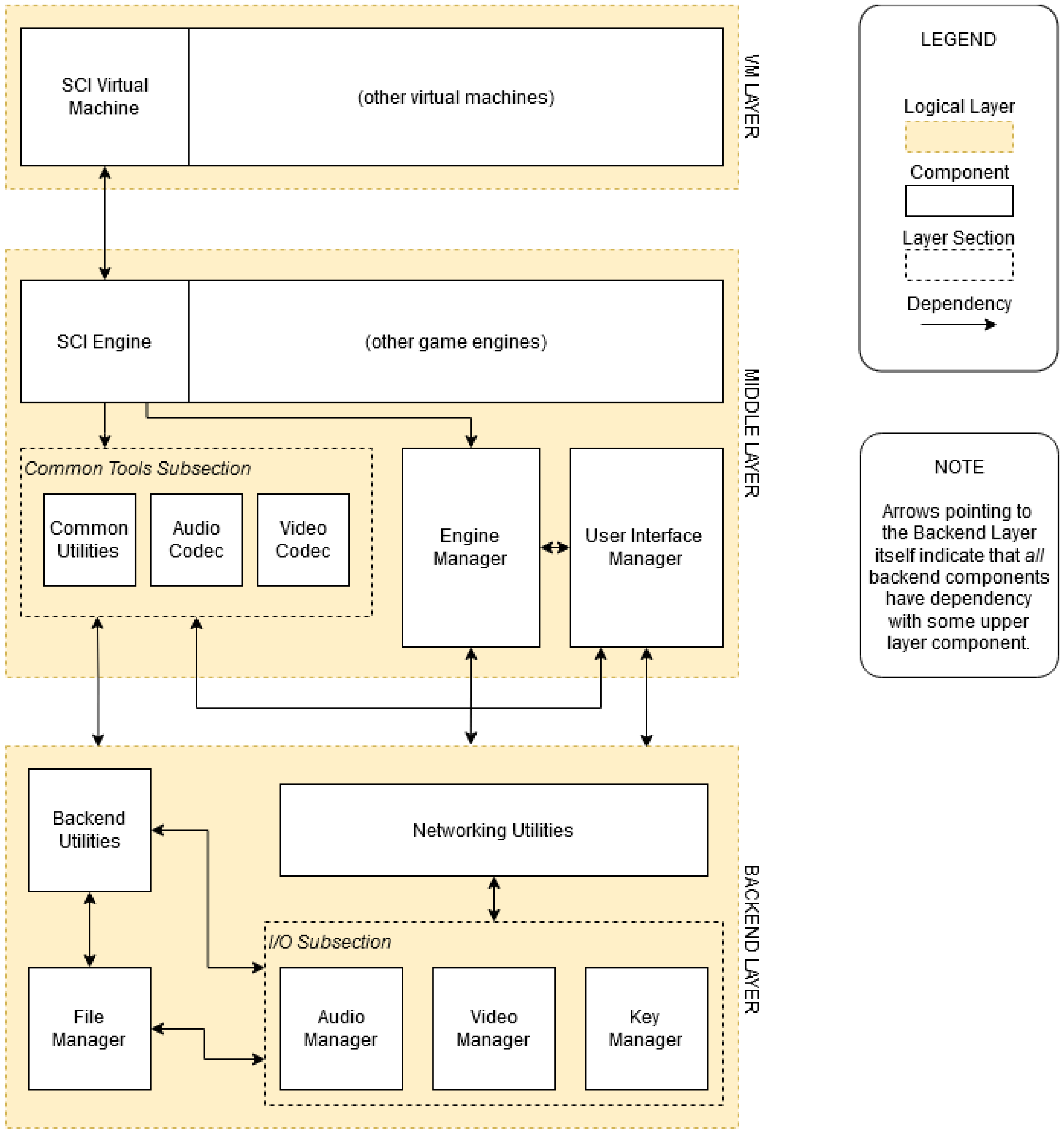


Fig I: Updated architecture diagram, reflecting the findings of A2, and our proposed enhancement.

DESIGNS FOR IMPLEMENTATION

Our architectural enhancement may potentially be implemented with one of two distinct structures. Each structure possesses the same fundamental components, however, they differ in the manner their components are separated and distributed amongst the greater client-server multiplayer system. Both of the below alternatives possess a number of primary internal components (see *Fig II/III* below on the next page). The 'Comm Manager' is the primary data node situated between the server implementation, the internet, and (potentially) the remainder of ScummVM. The Comm Manager utilizes a 'Packet Manager' which helps to seal and open cross-network data packets. In addition, the base ScummVM GUI will contain a sign-in interface. Alongside this addition, the Comm Manager will be used to verify incoming messages as from (or not from) valid ScummVM users.

Next, a subsection of the internal architecture relates to the physical server itself. This subsection contains an array of 'Player Instances' which each store data corresponding to a unique client connected to the server (i.e. player stats, position, name, etc). A component 'Server Common Data' will hold data/functions universal to all clients (i.e the game world). Finally, there exists a 'Session Manager' that detects alterations to the server state as a whole. The Session Manager acts in a publish-subscribe environment. Like an event bus, it will act to ensure proper synchronization of the server state across clients for all possible input messages the server may receive over time.

ALTERNATIVE I: "Host-Player System"

Our first design alternative involves an implementation of a hostable server directly within the ScummVM software downloaded by various users. In this design, all users have the ability to host their own game server from the specific machine and network they are running ScummVM on. A 'Host Game' button will be added to the starting GUI screen to initiate this process. Players will be able to remotely connect to this server from a new 'Join Game' starting screen button, where they will be able to input the server address of their desired host. The core concept of this alternative is the fact that the resulting multiplayer game will be executing on one single game engine (i.e. the host's), despite having input streams from multiple ScummVM instances (one for each player). Client players will have their inputs bundled into a packet and sent (via the client's and host's individual Comm Managers) into the host's own game engine (and server) to be processed. The resulting video output would then be concurrently sent back to each remotely connected player. For example, if a client player pressed a key to move their character, this keystroke would be sent (via the Comm Managers and internet) to the host's engine. The processed output (i.e. the character visibly moving on the screen) would be then transmitted to all players recorded within the server's Player Instance array (i.e. all clients view the motion).

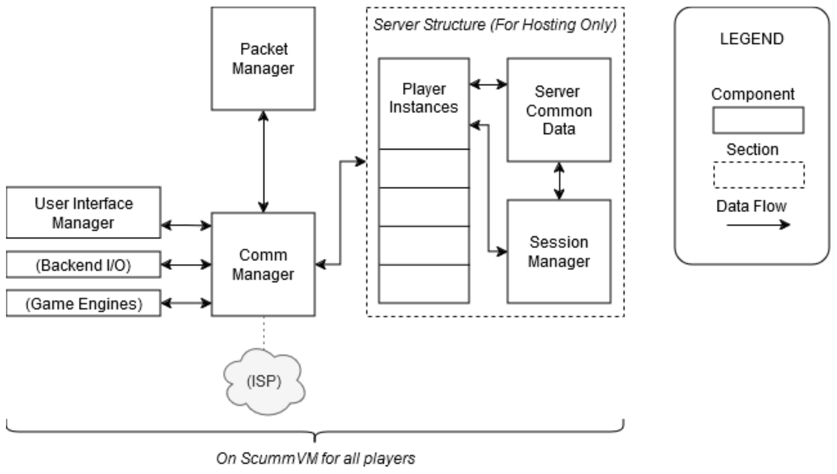


Fig II: Conceptual diagram for implementation Alternative I.

ALTERNATIVE II: Remote-Host System

Our second design idea contains the same core components as the prior alternative. However, it differs in how players communicate with the host, and the physical location of the host itself. Rather than having a designated host *player* responsible for all game engine and I/O processing, the server exists entirely separate from all players (i.e. there is *no* 'host player'). Instead, the server exists on some remote machine which works only to store and synchronize data across connected players. Thus, all connected players will make use of their own machine's game engine in processing input. The result of this processing would be then sent to the central server machine (i.e. the one not on any other player's device), which would then work to synchronize game output across all connected players. In short, this alternative differs from the first primarily in what data from the hosting Comm Manager is receiving as input. In contrast to the first alternative (where raw input data is received), the second alternative involves intake of data preprocessed by a certain player's own engine. Importantly, however, this system would require two distinct distribution options for the ScummVM software: one for clients (in which the main GUI page would *not* contain a 'Host' option), and one for the remote server (which essentially would *only* contain that option).

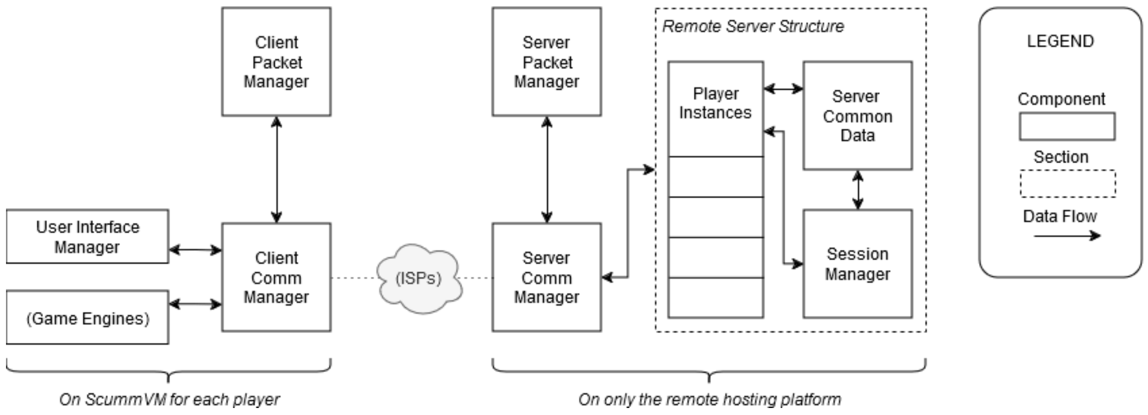


Fig III: Conceptual diagram for implementation Alternative II.

SEI SAAM ARCHITECTURAL ANALYSIS

Relevant stakeholders for this enhancement, alongside respective non-functional requirements:

Stakeholder	*Important NFRs*
Game Players	- *Usability*: players will need to connect to desired servers without any hassle. - *Performance*: once on a server, players need to have a gameplay experience as smooth and performative as if they were just playing locally. - *Availability*: servers should be able to run continuously, without players having to worry about the stability of their remote connections.
ScummVM Source Developers	- *Maintainability*: developers will want the networking system to be alterable due to ScummVM's open-source nature. - *Testability*: internet connection features may add a wide range of bugs during development that need to be fixed. - *Evolvability*: as operating systems themselves change over time, platform-specific ScummVM distributions (and thus any developers working below the middle layers) will need to effectively adapt.
Host Server Owners	- *Security*: hosts will want to ensure the users and data throughout their servers - *Performance*: hosts need to have high-performance servers to accommodate large numbers of player accesses - *Availability*: hosts want high availability to satisfy SLA requirements

Potential effects by each implementation alternative on the NFRs listed above:

NFR	*Effects of Alternative I*	*Effects of Alternative II*
Usability	Hosting will be generally an easier task, assuming the host player can successfully port-forward their server from their home network.	Since there is no specific 'host player', all players connected to the server will have equal concern over usability.
Performance	Due to the presence of only one working game engine (the hosts), the host player's machine and network will need to be comparatively stronger than all other client players.	Since this alternative uses multiple running game engines, the performance requirement is more distributed. All players will need to have machine/network strength, as opposed to *one* player needing a particularly powerful system.

Availability	Since all players can be a host, when any player hosted server goes down, it does not affect all players in the event of a failure to any single host. We can avoid service disruption across the system as a whole.	Most host providers provide a SLA. It is generally higher than the minimum availability requirements specified in the SLAs. But if service disruption it would be catastrophic.
Maintainability	A single centralized system with both client and server architectures will allow the code to be maintained without the hassle of two distinct distributions.	Distinct distributions for client and server mechanisms will result in the source code being less centralized. This may bring difficulty when maintaining components that rely heavily on direct S2C or C2S communications.
Testability	Since the server hosting architecture exists on the same distribution as the client, test suites/runs can more easily tie all networking features together.	Separate distributions for server/client systems will require more complicated testing structures for essentially the same tests.
Evolvability	The server architecture being more firmly rooted in the user distribution of ScummVM means that the server system will need to be made capable with more OS formats as a whole.	Having the server architecture based in another distribution will generally result in less machines having this architecture in the first place, so the demand for OS and environment changes will be lower.
Security	Because players can create their own host, players can have more direct control over the system, and thus threats to data integrity may be more easily circumvented.	Having entirely remote machines dedicated to hosting could potentially result in malicious (less-trustworthy) devices posing as public ScummVM servers.

FINAL IMPLEMENTATION DECISION

Based on the above features/details, Alternative 1 (host-player system) is more preferable for the ScummVM software, and its potential users/developers. The effects of usability in both cases is essentially negligible, as no alternative offers a ‘simpler’ *client* gameplay once the server is actually running. Availability/maintainability/security for the first alternative are superior, based on the above descriptions. Additionally, the performance requirements of Alternative 1 are not a major, concerning issue, as the games actually being run are not too resource-intensive. Ultimately, given the lightweight and open-source nature of ScummVM, the overall advantages of Alternative 2 are not comparatively strong.

EFFECTS ON HIGH LEVEL ARCHITECTURE

For high level conceptual architecture, after the implementation of the enhancement, the engine layer (which is inside the middle layer) would have lots of connection with this Networking Utilities component if there are new multiplayer games. The backend layer is affected too since it includes a networking directory, and we plan to add a ScummVM account system. The User Interface Manager is also affected because there will be some new buttons for being a host, joining a game and so on. For low-level conceptual architecture, the content will be discussed in the next part.

EFFECTS ON LOW LEVEL ARCHITECTURE

The following source code file directories may be influenced by our enhancement. Thus, they help define the effects of our enhancement on ScummVM's lower-level architecture.

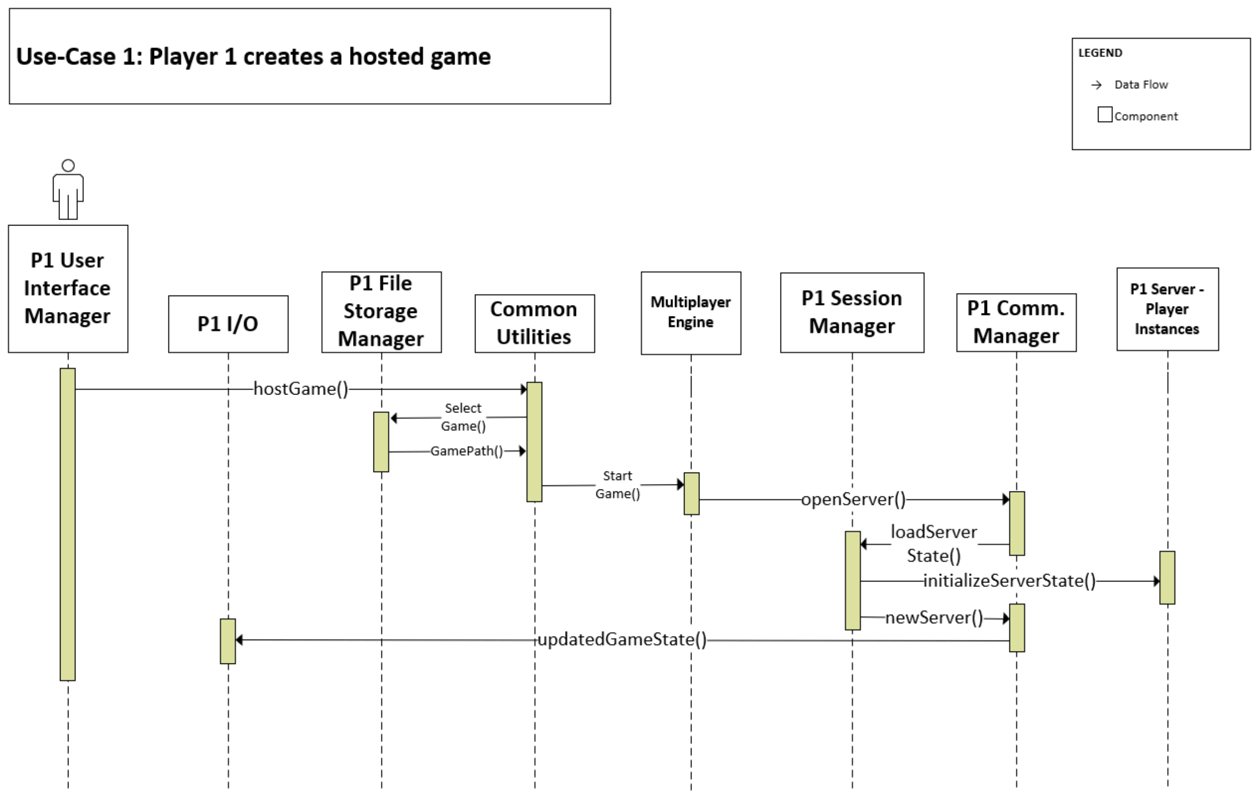
- *scummvm/backends/networking*. This directory includes functions about creation of hosts and making connections on the internet. They don't completely fit the purpose of Networking Utilities but will be used as help functions for the Networking Utilities.
- *scummvm/common* and *scummvm/gui*. These 2 directories contain lots of helper files like debugger.cpp, message.cpp and error.cpp that the new Networking Utilities would use for basic functions. What's more, since there are new buttons for the new functions. The GUI directory would add some new files.

POTENTIAL RISKS

One of the Potential risks is the high pressure on the server. When running a game with multiplayer, the enhancement is designed to only transform inputs and let the server deal with the inputs. The server will then send the results caused by the inputs back to every client for updating the game data, these would lead to strong pressure on the server. There's a possibility of bottlenecking the hardware dealing with too much data and there may be a risk of reducing connection quality or even crashing due to the heavy load.

Another potential factor that would cause a big impact is the quality of Player1's internet connection (the host in Alternative1). Since Player1 hosts the game and acts as a server to deal with all inputs from other players, any impact on the Internet to Player1 would directly affect other players.

SEQUENCE DIAGRAMS FOR USE CASES



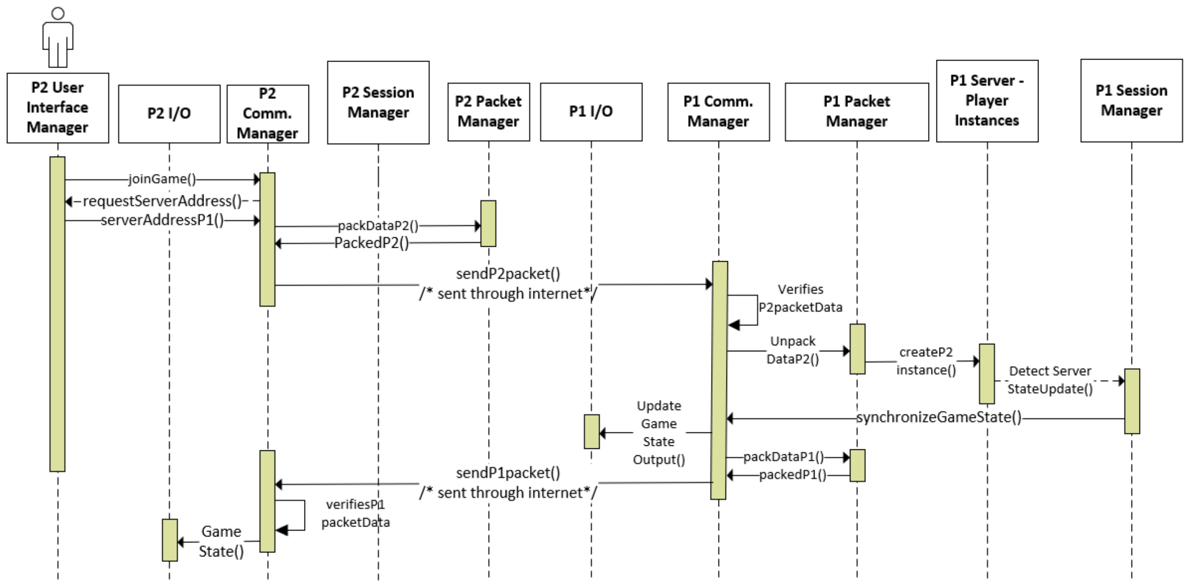
1. Player 1 clicks Host Game
2. Player 1 selects game they want to play sends call to Common Utilities
3. Common Utilities gets path from Player 1's File Manager
4. Common Utilities then sends message to Multiplayer Engine to start game
5. Multiplayer Engine requests Player 1's Communications Manager (henceforth Comm. Manager) to open a server for the player
6. Comm. Manager orders Session Manager to load a server state.
7. Session Manager initializes server state with single player instance.
8. Session Manager confirms server setup to Comm. Manager
9. Comm. Manager sends updated game state to I/O Subsection

Use-Case 2: Player 2 who is already signed in joins Player 1's hosted server

LEGEND

→ Data Flow

☐ Component



1. Player 2 clicks Join Game
2. Player 2 sends the server address of p1 to the Comm. Manager
3. Comm. Manager is going to use packet manager to send packet containing p2's user data (it uses PM to pack and unpack data)
4. Comm. Manager now has a packet of P2's user data and the address of P1
5. The packet is sent through the internet to P1's comm. manager
6. P1 then sends packet that came from P2 to its packet manager to unpack
7. P1's Comm. Manager verifies user identity and data
8. P1's Comm. Manager sends a message to server to create a new instance for player two
9. This triggers reply to the session manager (dotted line), which detects an update in the server state
10. Then the P1's Session Manager sends a request to P1's Comm. Manager to synchronize the server state across all player
11. Comm. Manager uses ScummVM to ensure proper output to display server state to P1
12. Comm. Manager then takes the same output that is being relayed to P1 and assembles a packet of this output using packet manager and P1's Comm. Manager sends the packet to P2's Comm. Manager through the internet.
13. P2 Comm. Manager receives packet from P1 and uses packet manager to unpack and update game state output

DERIVATION PROCESS

The process we went through to propose and realize the multiplayer feature was a lot of fun. We started by brainstorming among group members various features that would be cool. Many of the proposals were decided as not providing enough value to stakeholders. One example of a low value proposal was providing game developers a way to build a game within the ScummVM software itself rather than needing to fork the repository. Ultimately we decided that this proposal didn't offer enough value to demand a new feature. We landed on the multiplayer feature as the value was clear. The multiplayer feature offered players a way to connect with friends to play multiplayer games. Game developers had an opportunity to exercise their creativity to build games supported by this feature. The feature also had a positive value impact such as on the overall security requirements of ScummVM as it demanded these requirements in order to provide hosting services to its users.

Once we determined the feature we wanted to integrate into ScummVM, we proceeded to learn about how multiplayer games worked and gathered an understanding of how such a feature could be implemented. During this process, we discussed giving ScummVM players the ability to host their own game server. At this point, we had not yet discussed non-functional requirements, or the impact on multiple stakeholders. Our concern focussed on player experience. Once we were able to come up with a general understanding of how we could implement our feature, we considered an alternative. In the alternative, there is no 'host player'. Rather the server is independent of players and only works to synchronize data among all connected players. With the assistance of the SEI SAAM architectural analysis, we decided that our first proposed alternative would be more suitable for ScummVM. The analysis brought to light key non-functional requirements availability, maintainability, testability, and security that clearly favoured a model that gave players the ability to host their own game server. Each alternative had pros and cons but the benefits of alternative one ultimately outweighed the benefits of alternative two.

Upon deciding on our proposed feature implementation, we created two sequence diagrams that helped visualize the dependencies between ScummVM and our new feature. We ran into road blocks with lack of understanding of how data travelled across the internet. We learned about network communications, and how data travelled over the internet in packets.

We finished our proposal by discussing what new interactions we would need to test upon integrating our new feature with ScummVM. We discussed planning our testing of this feature and interactions by trying to understand at a high level the impact of the interactions. We considered various components of our new feature and the impact of interactions with the ScummVM system that require testing.

TESTING THE IMPACT OF INTERACTIONS

We planned to test the interactions between our new feature and the existing features by imagining what would happen if we made our own multiplayer game and hosted it on a ScummVM server. By doing this we would examine how the different components from the various layers interacted with each other and the new multiplayer framework. For example we would be able to determine, how seamlessly the game engine layer integrates with the new Networking Utilities component to handle tasks like player input, game state updates, and video output distribution; whether the common utility files like logging, messaging, and error handling are sufficient to support the networking-specific requirements; how the user-facing elements like the "Host Game" and "Join Game" UI components interact with the underlying networking functionality; any performance implications or scalability concerns when hosting a game server and managing multiple remote client connections; and edge cases around things like network failures, latency issues, or unexpected player behaviors, and how the system handles those situations.

LESSONS LEARNED

Throughout our process, we learned about the SEI SAAM architectural analysis and how it can be used to determine how to implement a new feature into a software. We also learned about how drastically new features can change how the software runs and how it changes the relationship between components. Finally, we learned about network communication, and how it can be used to transfer data between servers in order to implement a multi-player framework in ScummVM.

CONCLUSION

With our knowledge of the previous ScummVM architecture, we have proposed a possible improvement to the application. With the process of brainstorming ideas, we learned what ideas could work and what didn't, and we ultimately chose to add to the multiplayer framework. We looked through the possible interactions that this feature would bring to the system at both a high-level and low-level design.

We then analyzed potential stakeholders and the impact of non-functional requirements on these stakeholders. Building the sequence diagrams helped us visualize the game player experience and how data would flow between our new feature and the existing ScummVM system.

In conclusion, we have shown that we are capable of creating a new feature and how it can be applied to software. ScummVM, even though it is a relatively docile app that runs point and click adventure games, it’s interesting to see how it functions and how it is constructed.

TERM DEFINITIONS

API	Stands for "Application Programming Interface". This is a collection of functions/interfaces/etc that are used in order to facilitate communication between two distinct programs or software components.
OS	Stands for "Operating System", the foundation-level program on a machine managing the resources needed to run higher-level programs. The term "platform-specific" refers an implementation unique to a certain operating system (i.e. "ScummVM for Windows 10")
GUI	Stands for "Graphic User Interface". A directory inside the User Interface Manager component and provides a lot of helping functions that show messages on the screen like printing error messages.
Layered Architecture	An architectural style consisting of numerous stacked layers from top to bottom, in which each layer only has direct communication with those immediately above or below them.
C2S / S2C	"Client to Server" and "Server to Client", respectively. These abbreviations indicate that data is being moved from a client machine to a server machine at some other location.
SLA	A service level agreement (SLA) is a contract between a service provider and a customer that defines the service to be provided and the level of performance to be expected. An SLA also describes how performance will be measured and approved, and what happens if performance levels are not met.
ISP	The "Internet Service Provider" is the primary component a machine/software has when sending data/messages via internet communication. In the context of our enhancement, the Comm Managers for players/servers will use the device's ISP to facilitate sending packet data via the internet.

REFERENCES

ScummVM Developer Central. (n.d.). wiki.scummvm.org/index.php/Developer_Central.

ScummVM GitHub Repository. (n.d.). github.com/scummvm/scummvm.

Uurloon, Mic. “Design of a point and click adventure game engine”. *Groebelsloot*. 1 December 2015. www.groebelsloot.com/2015/12/01/design-of-a-point-and-click-adventure-game-engine/.

Source Multiplayer Networking. 12 August 2024. https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking

Using PlayFab Multiplayer Servers to host multiplayer games. 21 April 2023. https://learn.microsoft.com/en-us/gaming/playfab/features/multiplayer/servers/using-playfab-servers-to-host-games#6-connect-and-play

What is an SLA (service level agreement)?-IBM. 30 May 2024. https://www.ibm.com/topics/service-level-agreement