6.0 Architectural Design Document

6.1 Introduction

This document presents the architecture design for the software comprising the project *Cosmonarium.* The project offers users a turn-based role-playing computer game that draws upon themes and styles of the JRPGS developed by professional video game companies in the 1990s. Built using the lightweight Godot Engine (ver. 3.2.2), *Cosmonarium* integrates story, dialogue, graphical, musical, and other audio elements produced in auxiliary software with game logic in order to create a complete playable product.

6.1.1 System Objectives

The objective of *Cosmonarium* is to offer users an entertaining, thought-provoking, and nostalgic gaming experience. The project seeks to provide new and experienced gamers alike with intuitive controls, an engaging storyline, difficult yet balanced battles against computer-controlled enemies, interesting scenes for exploration, clever dialogue with non-player characters, and a lovingly crafted soundtrack. As such, the game ought to fully seize players' attention and many of their senses. Another goal that the project aims to fulfill is to create a game that keeps players' other time commitments in mind. Many modern RPGs require an investment of 60+ hours in order for players to make real progress. In comparison, *Cosmonairum* aims to be much shorter, requiring only several hours for players to finish its main storyline. This shorter time for completion should make *Cosmonarum* more accessible to players who are either new to the RPG genre or who are unfortunately too busy with their daily lives to frequently play games.

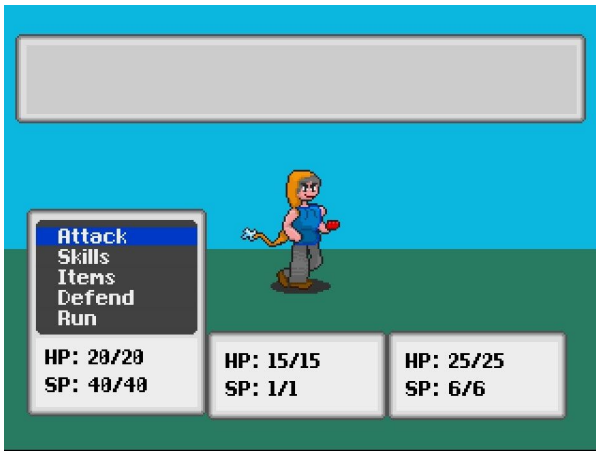
6.1.2 Hardware, Software and Human Interfaces

6.1.2.1 Graphical User Interface

The game environment will be displayed to the user via a GUI. The shown game environment will differ drastically depending on whether the player is currently in exploration mode or combat mode. In exploration mode, a user will usually see the player character centered in the middle of her or his screen, followed by any other characters in the party. As the player moves around (using either a keyboard or controller), the camera remains focused on the main character. Note that the camera may pan away from the main character during certain sequenced cutscenes. Non-player characters, enemies, dialogue boxes, interactable item boxes, and the background scenery will also be visible in relation to the party during exploration. During combat mode, the GUI can be separated into four distinct parts. The top of the screen shows a user any textual battle information based on the events occurring during battle, e.g. whether a character has attacked or taken damage. The central row of the battle GUI contains the enemy sprites corresponding to the foes that the player is currently facing in combat. Finally, roughly

the bottom third of the screen displays the player character modules. Each of these modules presents each character's current battle stats and provides a popup menu that allows a user to select which actions to take in battle. These three sections of the screen are backed by a colorful backdrop. Example GUIs for both exploration and battle mode may be seen below.

Exploration Mode



Battle Mode

6.1.2.2.1 Keyboard

The players will primarily interact with the game characters and environments of *Cosmonarium* via a keyboard. A compatible keyboard may be either built-in to the machine running the game (e.g. on a laptop computer) or it may be connected to the machine through USB insertion or another hardware interfacing method (e.g. with a desktop computer). Users may affect what happens in the game environment by pressing or holding certain keys. Players may move their characters in the overworld using the arrow keys; menus may also be navigated in this fashion. Another set of specified keys will allow users to interact with non-player characters and certain game objects, open up and close menus, progress dialogue, and select, accept, or reject certain options (e.g. which item to use or which choice to make in a battle) as displayed on-screen. *Cosmonarium* launches initially with a default key mapping for player controls, but there will be an option for users to optionally modify which keys are mapped to which commands. The keyboard may also be utilized to input text at the outset of a new game when the user has the chance to rename the main characters.

6.1.2.2.2 PC Gaming Controller

Alternatively, users may interact with the game environment via any compatible external video game controller. Such a controller must be joined to the desktop or laptop computer running *Cosmonarium* either by USB insertion or by a wireless Bluetooth connection. Allowing users to interface with the game through such controllers aims to increase the retro nostalgia that users

may experience. All of the input commands a user may provide on a keyboard have their equivalents on a compatible controller. Typically, character movement and menu traversal commands shall be mapped to the linked controller's directional pad (also called the d-pad), and inputs used for interfacing with menu options and dialogue boxes shall be mapped to the face buttons (generally A, B, X, and Y). The project plans to give players the option to customize the button mappings on their controllers to a certain degree, though the default configuration is highly recommended.

6.1.2.3 Mouse

A mouse or trackpad may be necessary for initially accessing the game file(s). Depending on how *Cosmonarium* is distributed following development and its successful release, users may want a mouse or its equivalent to navigate through web pages or file directories. All of these steps will take place prior to the launch of the actual project software. However, it should be noted that the inner gamespace of *Cosmonarium* will not support mouse controls. Menu selections and character movement shall all be handled by the inputs that the player provides to either the keyboard or gaming controller that she or he is using. This decision to prevent mouse usage in-game stems from the project goal of recreating the 1990s RPG play style as much as possible.

6.1.2.4 Computer Speakers (or Similar Audio System)

Although not a requirement for running *Cosmonarium*, having built-in computer speakers or a sound system connected to the machine running the game will enhance the overall user experience. Players will be able to hear the game's music and sound effects via an adequate audio system.

6.2 Architectural Design

The software architecture for *Cosmonarium* can be broadly divided up into three major subsystems: User Input Handling, Persistent Data Handling, and Game Environment Management. The first two subsystems are crucial for the game to operate properly, while the third subsystem deals with a majority of the inner game logic. As such, the Game Environment Management subsystem is further broken down into smaller, more specific modules.

6.2.1 Major Software Components

6.2.1.1 User Input Handling

The User Input Handling subsystem deals with converting all user inputs for the player characters in-game. This subsystem is responsible for accepting key or button presses, interpreting their meaning in the context of the current game mode, and executing the related commands. In general, this would refer to player character movement in exploration mode and selecting menu options in battle mode. Any inputs that are not mapped to a specific key or button

in the current configuration are ignored. This subsystem primarily consists of an Input Engine and related character scripts and objects.

6.2.1.1.1 Input Engine

The Input Engine performs the conversions necessary to translate user inputs to character actions. This minor subsystem facilitates much of the player's ability to interact with the game environment.

6.2.1.2 Persistent Data Handling

The Persistent Data Handling subsystem keeps track of users' save and load data. This responsibility includes providing load files in conjunction with *Cosmonarium*'s title screen and permitting users to save their current game progress during particular sections of exploration mode. This subsystem must ensure that players may save, leave the game for an unspecified amount of time, return and resume their gameplay from the point at which they last saved. This subsystem primarily consists of a Save Manager and ways of storing data persistently.

6.2.1.2.1 Save Manager

The Save Manager handles the collecting, saving and loading of persistent data for users. This subsystem works with the user machine's file system to keep saved game data safe.

6.2.1.3 Game Environment Management

The Game Environment Management subsystem controls a majority of the action and on-screen changes displayed to the user via the GUI. In a nutshell, this subsystem manages all the pieces of the project that do not rely directly on user input or save data. Since this is a fairly vague description, a bit more context for the Game Environment Management system is provided by the subsequent list of minor subsystems. Each of these minor subsystems will be further explained in the detailed design section.

6.2.1.3.1 Scene Manager

The Scene Manager smoothly transitions between established scenes in the game environment. This minor subsystem switches between scenes in exploration mode, and also handles swapping between exploration and battle scenes.

6.2.1.3.2 Menu Manager

The Menu Manager controls the layering of UI submenu scenes that allow the user to interact with usable persistent game objects as well as information of the state of playable characters and general game environment configuration.

6.2.1.3.3 Camera Manager

The Camera Manager handles what the user can perceive on-screen at any given time. This minor subsystem determines how much of the game environment the user sees during exploration mode based on the user's window size and screen resolution.

6.2.1.3.4 Dialogue Engine

The Dialogue Engine hooks up pre-scripted dialogue to certain non-player characters and associated cutscenes. This minor subsystem is responsible for displaying dialogue to the player in its correct order and at an appropriate speed for easy reading.

6.2.1.3.5 Background Audio Engine

The Background Audio Engine determines which integrated music piece plays, depending on the current mode of the game environment. This minor subsystem swaps the playing audio files between exploration areas and battle scenes as well as during specified cutscenes.

6.2.1.3.6 Sequencer

The Sequencer executes the commands provided by an external script to build cutscenes. This minor subsystem takes care of launching any and all events that may occur during a scripted cutscene.

6.2.1.3.7 Actor Engine

The Actor Engine executes sequenced commands for the given actors . This minor subsystem directs actors (player characters, unique non-player characters and enemies, objects, etc.) to perform the actions specified by the provided commands.

6.2.1.3.8 Enemy Handler

The Enemy Handler deals with how and where enemy characters are spawned during exploration. This minor subsystem affects how many enemies can be in the scene at one time, and keeps track of enemy data to pass to battle scenes.

6.2.2 Major Software Interactions

6.2.2.0 *Prior Auxiliary Software Imports

During development and prior to the actual execution of the final game files by end users, *Cosmonarium*'s core components in the Godot Engine interact with auxiliary software to import graphical, dialogue, musical, and audio assets. Specifically, Pro Motion NG will pass in files for character sprites, Metalogue will provide dialogue support, and Famitracker shall add in files for music playback. Sound effects will be pulled from open source asset folders or generated on a case-by-case basis.

6.2.2.1 User Input Handling ↔ Game Environment Management (General case)

When a user inputs key or button presses, the User Input Handling subsystem parses these inputs and translates them to mapped commands.. In response, the main character sprite will move and act according to these inputs within the game environment. Based on the player's movement, the Game Environment Management subsystem may spark reactionary changes in the environment, such as triggering a cutscene or causing an enemy to spawn outside of the player's view. Ideally, these two subsystems would continuously interact with one another for the duration of any gaming session.

6.2.2.2 Game Environment Management ↔ Persistent Data Handling

At certain locations within the game environment, a user shall be able to save her or his progress through the story. The Game Environment Management subsystem will pass the persistent data to the Persistent Data Handling subsystem. This latter subsystem will then ensure that the game data is properly stored on the user's machine. What this persistent data consists of will be discussed further in the corresponding detailed design section. Moving in the opposite direction, the Persistent Data Handling subsystem would load the correct save file when a user returns to a saved game session. The loaded persistent data would be passed back to the Game Environment Management subsystem, and the player party and surrounding environment would be reinstated based on whatever information was provided.

6.2.2.3 Game Environment Management ↔ Scene Manager

Based on a player's actions within the game environment, the Scene Manager may be signaled to change out the current scene for a different one, modifying the environment in the process.

6.2.2.4 Scene Manager ↔ Enemy Handler

The Scene Manager communicates with the Enemy Handler to decide how, where, which, and how many enemies may be instanced into the current scene.

6.2.2.5 Scene Manager↔Menu Manager

The Scene Manager does not interact with the Menu Manager because by design the Menu Manager is meant to work within an exploration scene without needing to reload the game state.

6.2.2.6 Scene Manager → Background Audio Engine

The Scene Manager instructs the Background Audio Engine to play back the soundtrack file associated with the current scene or exploration area.

6.2.2.7 Game Environment Management ↔ Sequencer

Based on a player's actions within the game environment, the Sequencer may be signaled to queue up and execute a pre-scripted cutscene that plays within the environment. During any one of these cutscenes, the main character and party, specified enemies and non-player characters, and movable objects may be moved and placed around the environment. These changes may or

may not persist immediately after the cutscene has concluded; such behavior will be determined on a case-by-case basis.

6.2.2.8 Sequencer → User Input Handling

Following any point in which the Sequencer has been signaled to execute a cutscene, the Sequencer will direct the User Input Handling subsystem to block the interpretation of any user input. Thus, the player will be unable to control their character during sequenced events. The Sequencer shall return control after the completion of its final instruction in that specific cutscene.

6.2.2.9 Sequencer ↔ Actor Engine

Following any point in which the Sequencer has been signaled to execute a cutscene, the Sequencer may direct the Actor Engine to call its own command on an actor within the scene. After calling the specified actor command, the Actor Engine will report back to the Sequencer that the next sequenced instruction can be loaded and executed.

6.2.2.10 Sequencer → Background Audio Engine

Following any point in which the Sequencer has been signaled to execute a cutscene, the Sequencer may direct the Background Audio Engine to switch to a different song in the soundtrack or to stop playback entirely.

6.2.2.11 Sequencer ↔ Camera Manager

Following any point in which the Sequencer has been signaled to execute a cutscene, the Sequencer may direct the Camera Manager to move the camera position to another specified location within the scene. After completing this move, the Camera Manager reports back to the Sequencer so that the next sequenced instruction can be loaded and executed.

6.2.2.12 Sequencer ↔ Dialogue Engine

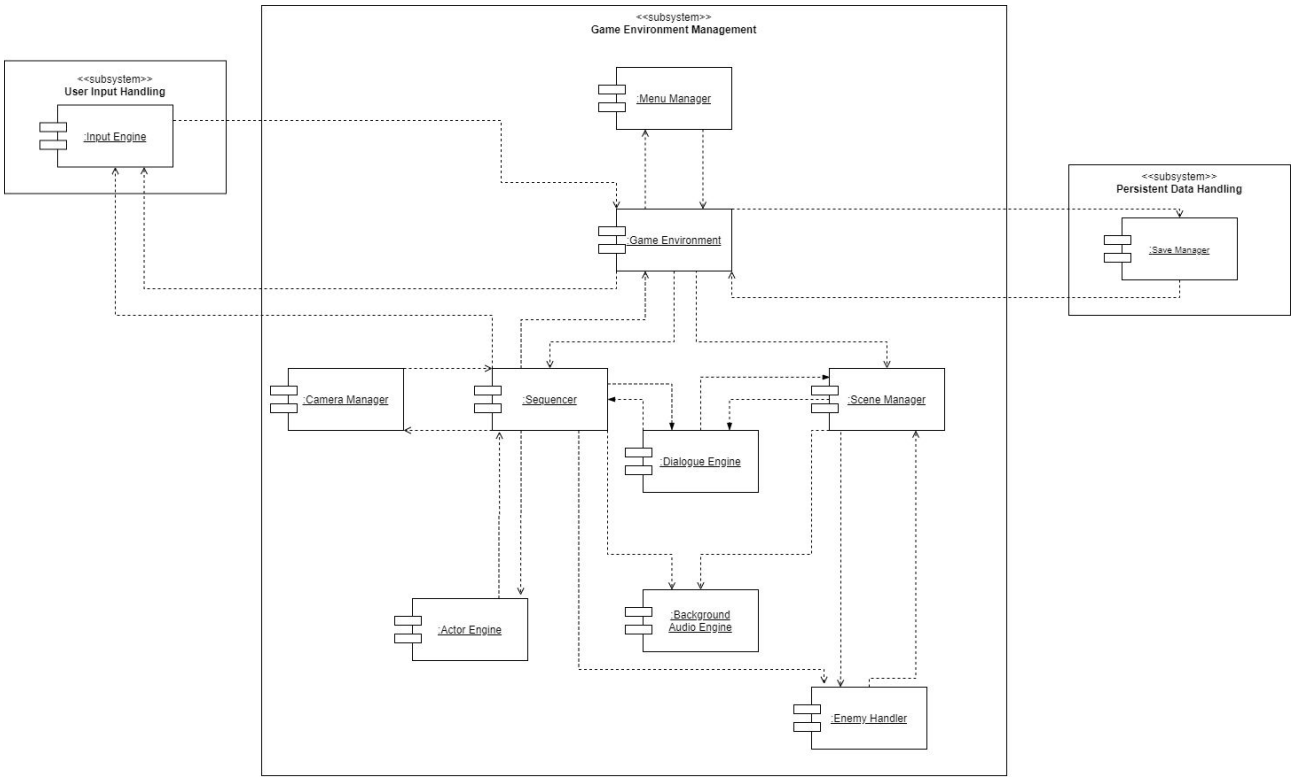
Following any point in which the Sequencer has been signaled to execute a cutscene, the Sequencer may ask the Dialogue Engine for a specific piece of scripted dialogue to be displayed to the user. The Dialogue Engine presents this dialogue visually, then reports back to the Sequencer so that the next sequenced instruction can be loaded and executed.

6.2.2.13 Sequencer → Enemy Handler

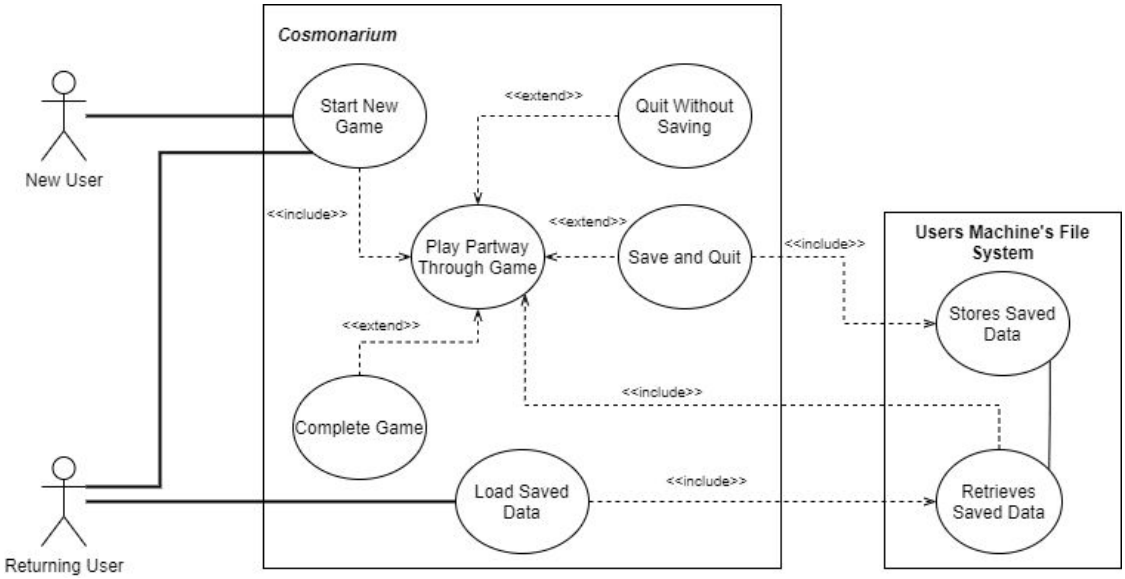
Following any point in which the Sequencer has been signaled to execute a cutscene, the Sequencer directs the Enemy Handler to prevent enemies from spawning during the cutscene. The Sequencer shall return control over spawning behavior to the Enemy Handler after the completion of its final instruction in that specific cutscene.

6.2.3 Architectural Design Diagrams

6.2.3.1 Component Diagram



6.2.3.2 Use Case Diagram



6.2.3.3 State Diagram