Software Requirements Specifications

Firmware Infirmary

Version 2.1

Defined By:

Billy Carroll, Jovan Kouakou, Mike Mariano, Nick Pelletier, Ryan Marcovecchio, Tom Card

Advisor:

Jeff Salvage

Stakeholder:

Drexel University

Table of Contents

1. Introduction

1.1. Purpose

This Software Requirements Specification (SRS) is a comprehensive document that outlines the requirements for the development of a new game, "Firmware Infirmary". It outlines the game, its development, its framework, and its overall future improvement.

1.2. Overview

The game is a narrative-based simulation game that puts players in the role of a mechanic working in a near future environment to repair cybernetic attachments for customers. The game combines elements of customer interaction through a dialogue system, diagnostic procedures, and activities to create a unique and engaging experience for players.

The primary objective of the game is to provide players with an immersive, action-packed experience that places them in the world of transhumanism and the politics of disease. The game centers around a mysterious disease that suddenly appears and devastates the community of performers with extra-anatomical, cybernetic attachments. Players work to uncover the cause of the disease and find a way to save their community, while balancing the care of their customers and the politics of the powers that be.

Firmware Infirmary features an activity system that interacts with the narrative in two ways, either through positive and negative results or through contextual dressing. The game also features a multi-year time jump in the middle of the game that allows players to see the progression of the disease in the world and characters, and follow the threads of their decisions to their distant conclusions.

1.3. Scope

This document outlines the design and development framework for the game Firmware Infirmary. The release of this game will follow the outline of specifications given by GDAP's professor, Rob Lloyd. Programmers use this document to understand the innards of the system, and testers will use this to verify the system works as intended. End-users can also utilize this document to better understand the inner workings of the game.

1.4. Definition

1.4.1. **Collider**: A component in Unity that defines an object's physical shape and allows it to interact with other colliders in the game world.

1.4.2. **Coroutine**: A type of function in Unity that allows code to be executed over multiple frames, useful for animating or updating objects over time.

1.4.3. **HUD (Heads-Up Display)**: A type of user interface that displays information to the player on top of the game screen, without obscuring the view of the game world.

1.4.4. **Interactable**: A term used in game development to describe an object that can be interacted with by the player, typically through the use of buttons or other input methods.

1.4.5. **LineRender**: A component in Unity that allows developers to draw lines and shapes on the screen, typically used for visual effects or debugging.

1.4.6. **NPC (Non-Player Character)**: A character in a game that is controlled by the computer, rather than by the player

1.4.7. **Prefab**: A reusable game object template in Unity, typically used for creating multiple instances of the same object with the same properties.

1.4.8. **Raycast**: A technique used in game development to detect if an object is hit by a virtual ray, typically used for line of sight or aim detection.

1.4.9. **Rigidbody**: A component in Unity that allows an object to be affected by physics and interact with other objects in the game world.

1.4.10. **SRS (Software Requirements Specification)**: A document that outlines the functional and nonfunctional requirements for a software system.

1.4.11. **Trigger**: A type of collision detection in Unity that occurs when two colliders come into contact, typically used for triggering events or interactions.

1.4.12. **UI (User Interface)**: The part of a software system that interacts with the user, allowing them to input commands and receive feedback.

1.4.13. **VFX (Visual Effects)**: Images created to enhance and simulate effects to create an enhanced experience.

2. Functional Requirements

2.1. Controls

2.1.1. Default Controls

Input	Output
W	Player Walks Forward
A	Player Walks to the Left
S	Player Walks Backwards
D	Player Walks to the Right
Left Mouse Button (LftMb)	Interact
Escape (Esc)	Pause/Unpause

2.1.2. Activity and RingCamera Controls.

Input	Output
Left Click	Grab Object
Right Click	Drop Object (in upgrades games)
Space Bar	Skip Dialogue

2.2. Gameplay Loop

2.2.1. Day Cycle

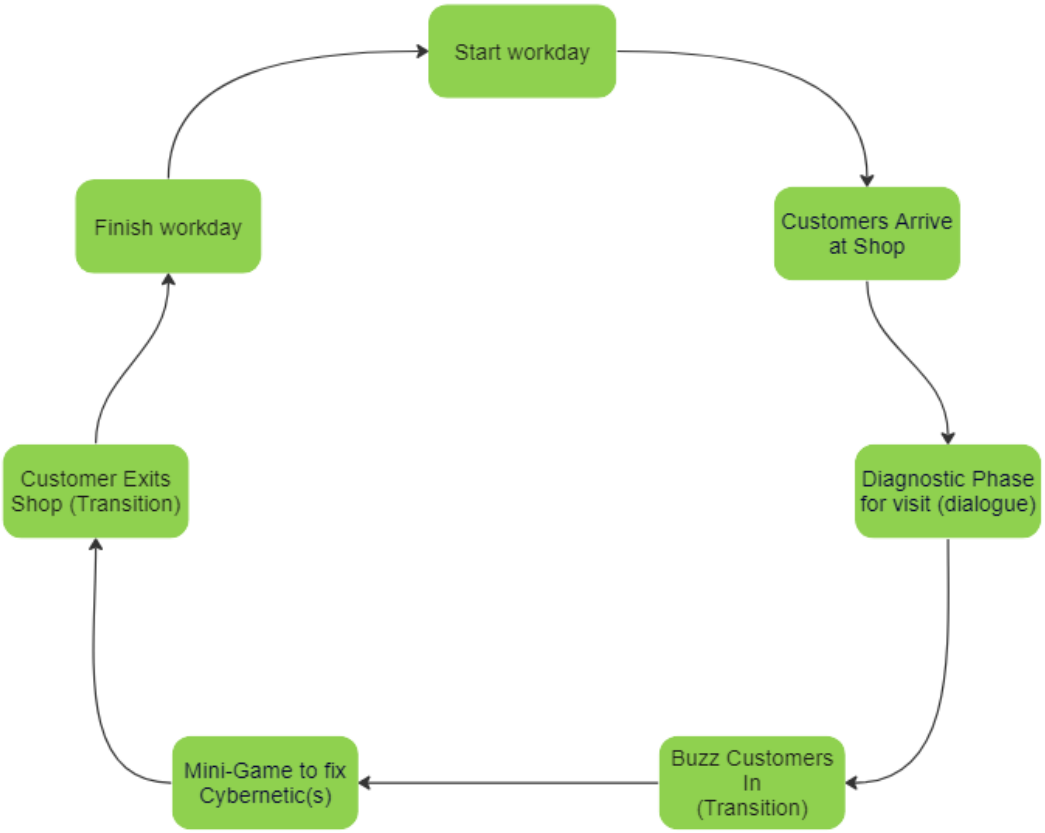
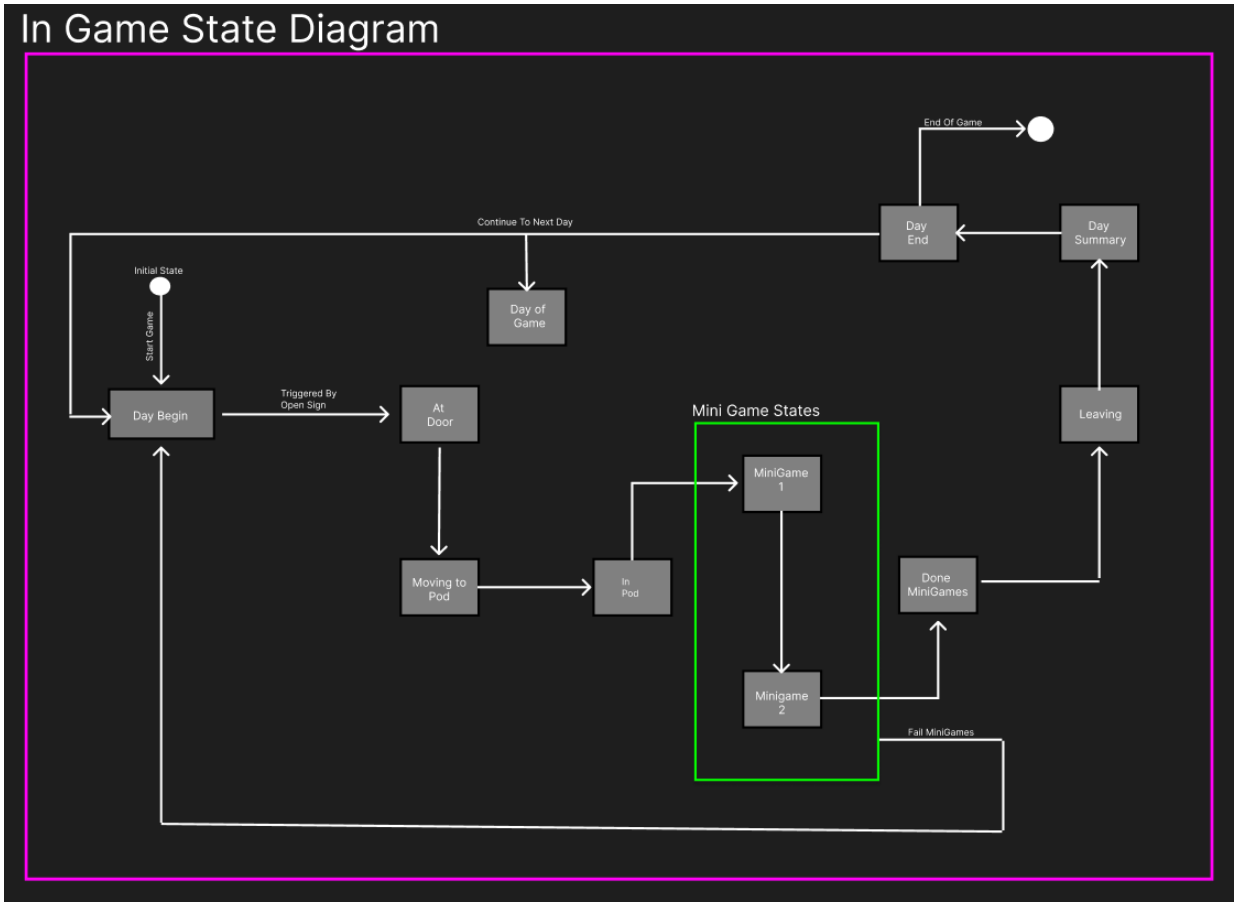


Figure 2.1: Diagram displaying overarching day loop



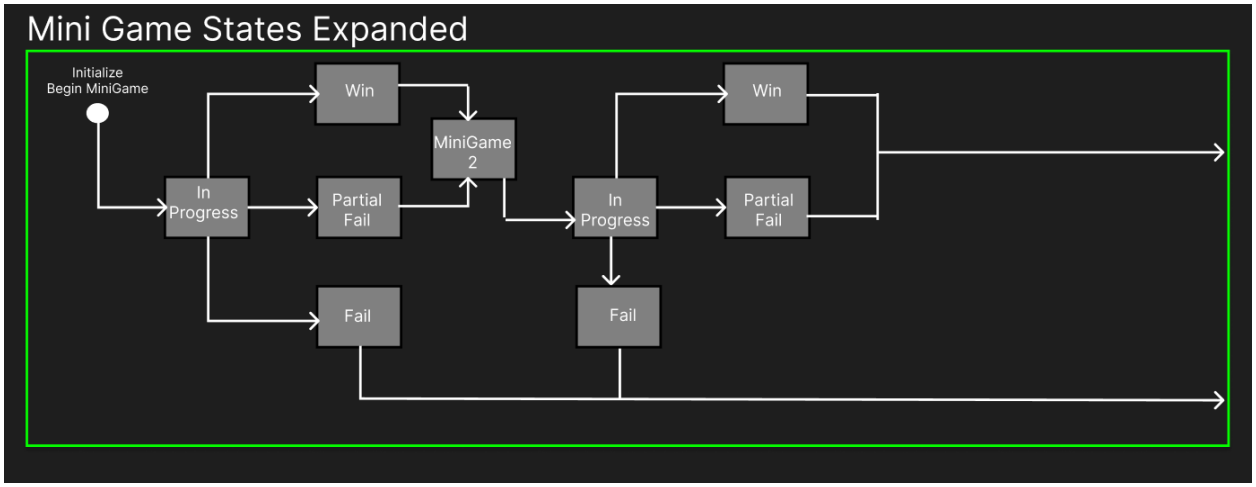


Figure 2.2: Diagram displaying day cycle state machine

2.2.1.1. Start Workday

A. Players walk around the shop using WASD

B. Players interact with the lever by using the Left Mouse Button

C. After 5 seconds, a customer arrives at the shop, indicated via the intercom system through a light and audio queue as well as an ‘OPEN’ sign turning on.
D. Light is a small shop light on the wall that blinks once a customer arrives
E. A phone ringing/beeping is the audio cue once a customer arrives

2.2.1.2. Customer Arrives

A. Customers can only arrive after the ‘OPEN’ sign is enabled
B. An audio cue and light indicate that someone is interacting with the intercom system
C. Players interact with the intercom system by using the Left Mouse Button on the intercom box.

2.2.1.3. Diagnostic Phase

A. Players enable the intercom system screen, which is a 2D “video feed” of the customer
B. Players interact with the dialogue system, asking brief questions prompted to the customer. Once all information is gathered and the conversation is complete, the player’s screen exits the intercom system
C. Intercom system experience should take 45-60 seconds of gameplay depending on the length of dialogue

2.2.1.4. Service Chamber

A. Gameplay scene transition (fade in)
B. Player clicks buttons on the left hand side of the patient tube that the patient is now in.
C. Player enters the first activity.

2.2.1.5. Activity Fixes

A. 2 activities per customer (activity details below)
B. 5 minute long gameplay (max) per activity

2.2.1.6. Customer Exits

A. Transition sequence

2.2.1.7. End Workday

A. Players interact with the computer to reach the next day.

2.3. Activities

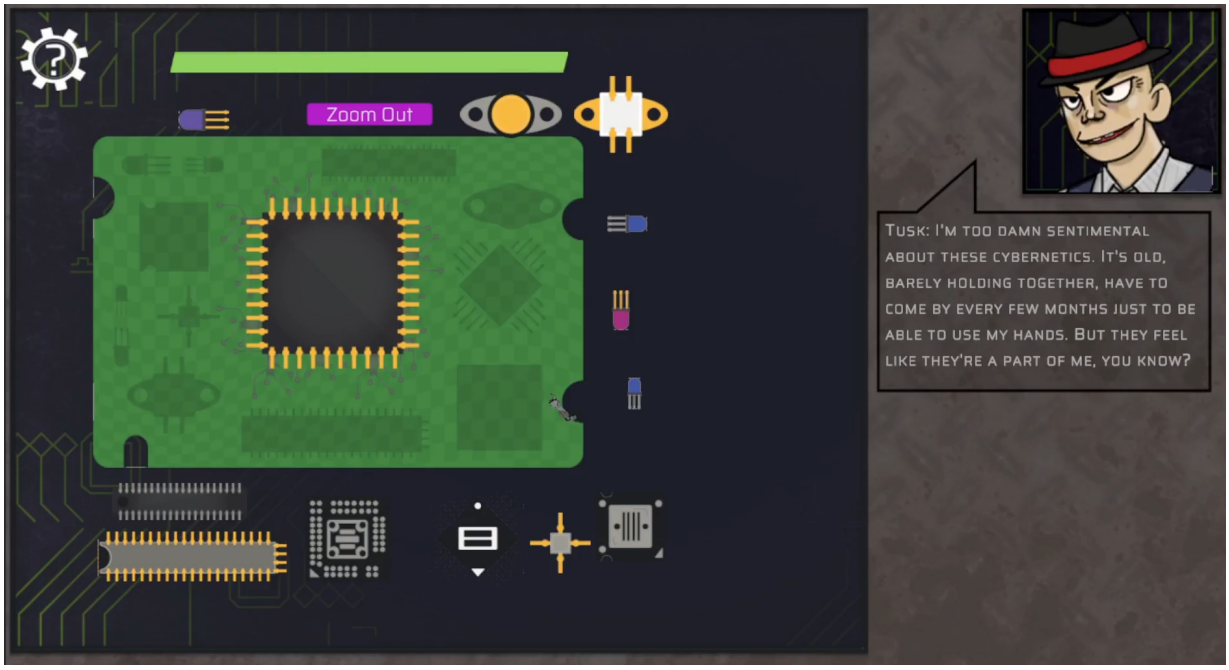
2.3.1. Repair Broken Devices

The UI opens to full-screen. The player interacts with dialogue options and a cybernetic relevant to the customer. The user must select the outlined section of the object to start the activity. To complete the activity the user must move all of the floating parts in the interface back into their respective outlines. They must also complete the dialogue in which they talk with the patient. There are win and loss states based on the timer at the top of the UI.

Figure 2.3: The cybernetic part (shown on the left) and the dialogue interface (right)

2.3.1.1. Dialogue

A. There are 0 to 4 ***Dialogue*** options.
B. Clicking on a ***Dialogue*** option progresses the conversation.
    a. Dialogue ends or stops to wait for the player to finish part of the activity.



2.3.1.2. GamePlay

A. Clicking the highlighted section of the cybernetic opens up the repair section of the game.
    a. Clicking and dragging a component over an outline snaps it to it.
    b. A component is in place when it is in the correct outline that matches its shape.

Win Condition

When the user moves all of the components into place with time remaining on the timer a win condition is met. The win condition is displayed and the Exit button shows.

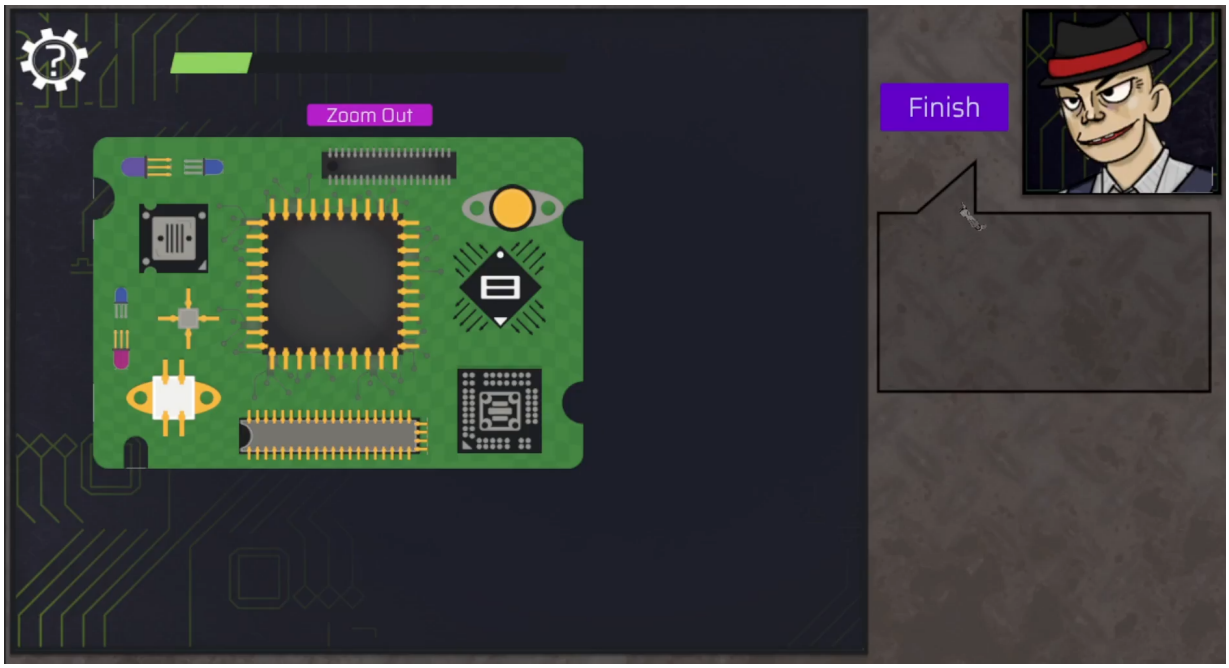
Partial Lose Condition

A minor loss condition is met when all pieces are put into place, but in incorrect places The loss dialogue displays and the Exit button shows.

Lose Condition

A major loss condition is met when the timer hits 0 at any point during play.

The player is notified of failure, and the day is restarted



2.3.2. Paint Job

A. The player selects from a group of 5 colors to correctly match the current color with the desired color shown on the left-hand side of the game. By clicking on the different color splotches on the right-hand side, the player creates the desired color. If the current color matches the desired color when the player clicks on the accept color button, the player advances to the next portion of the activity; if not the game does not let the player progress (Figure 2.6).

B. The player is presented with an object to paint and the player clicks and holds to paint where they choose. They paint on the object, resulting in a better score, or off the object, resulting in a worse score. When the player finishes painting or when they run out of paint, as shown by the paint meter on the left-most side of the screen, they click the accept paint button and end the activity (Figure 2.7).

C. During both of these portions of the activity the player chooses dialogue options and when both the dialogue options are exhausted and the activity is complete, the activity is complete.

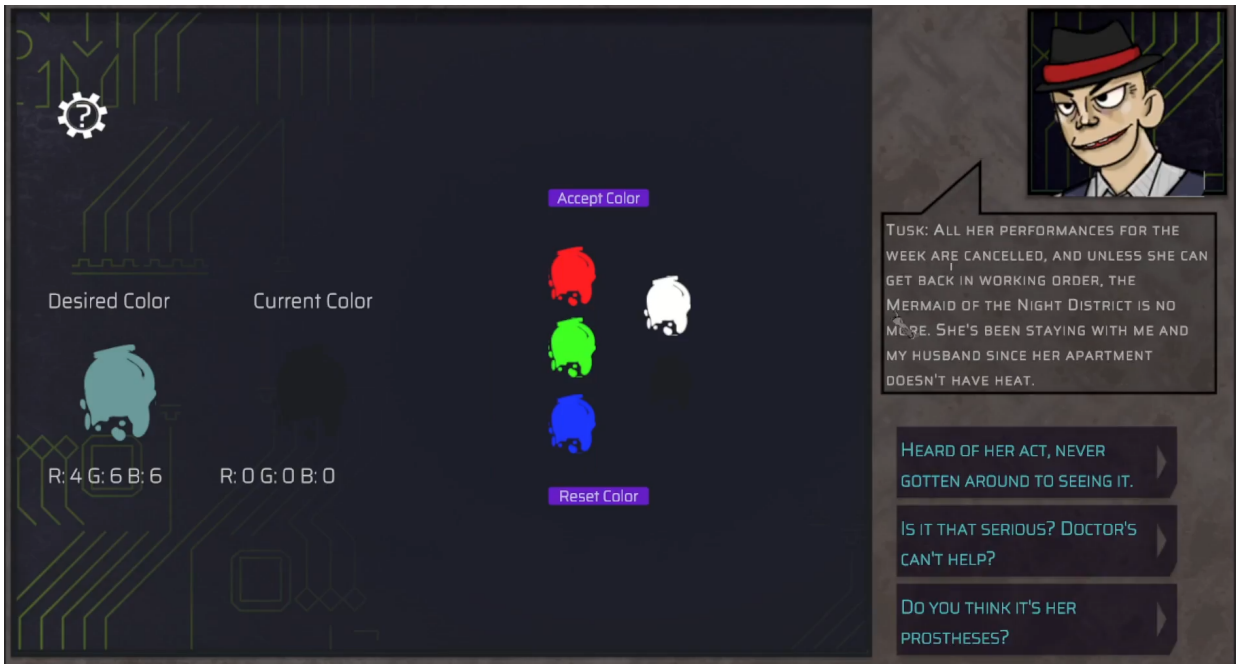


Figure 2.6

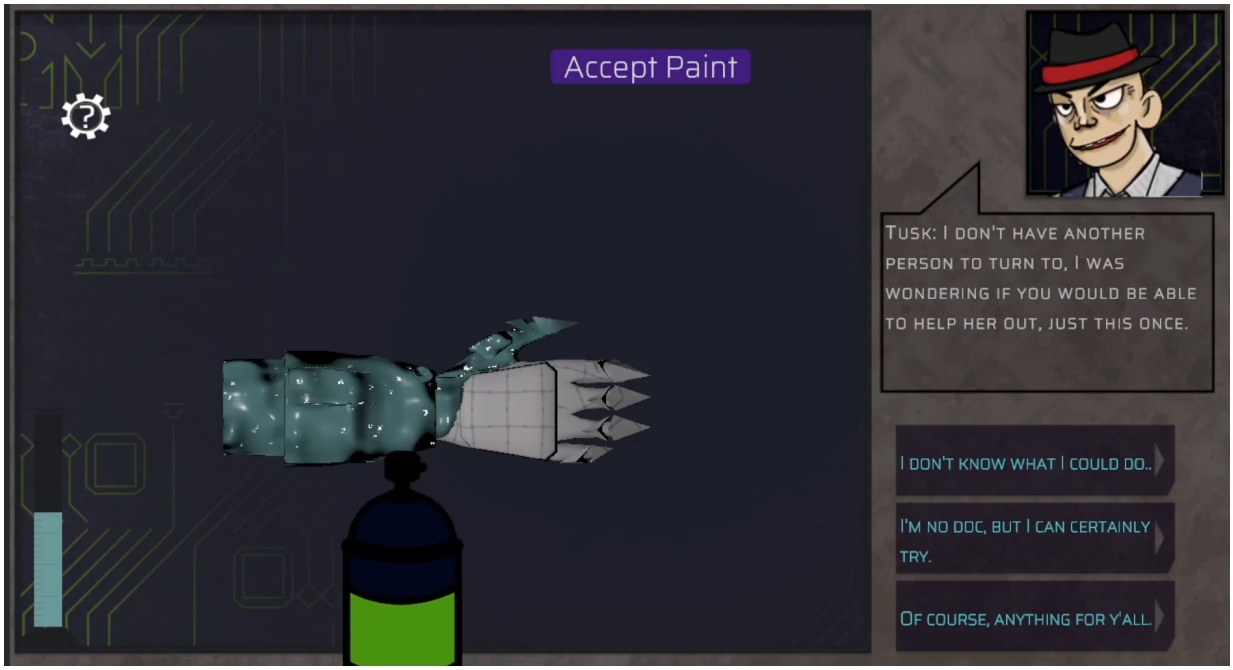


Figure 2.7

2.3.2.1. Dialogue

A. Clicking a Dialogue option progresses the dialogue.
    a. Dialogue ends at some point or stops to wait for the player to finish part of the activity.

Win Condition

The user finishes spraying within the lines according to the customer's asking. Depending on how well they do the job (how much paint was left), there are different dialogues after completing the job(s).

Lose Condition

There is no lose condition.

2.3.3. Upgrades, People, Upgrades

A layer-based puzzle made of 3D assets. The focus is on disassembling the customer's cybernetic and adding a new part all without touching any area outside of the upgrade space. Players have a choice of tools and must use

the correct tool in order. An additional menu is available in the gameplay area displaying the removed pieces for their replacement after the upgraded part is installed. A meter measures the amount of static shock generated by misclicking (Figure 2.8).

Figure 2.8

2.3.3.1. Dialogue

A. Clicking a Dialogue option progresses the dialogue.
B. Dialogue ends at some point or stops to wait for the player to finish part of the activity.

2.3.3.2. Gameplay

A. Select the correct tool for the upgrade and remove internal parts of the cybernetic in order to reach the part that needs upgrading or replacement.
B. In some cases, the part itself is all that needs removal. In other cases, the player must move additional parts out of the way (usually using a tool) to reach the part that needs upgrading.
C. After the upgrade, the player places all internal parts back in the cybernetic.
D. Clicking on something that's not a component reduces the green cybernetic stability bar

Win Condition

The user completes the upgrades without the cybernetic stability dropping below 40%

Partial Lose Condition

The user completes the upgrades but the cybernetic stability is 40% or below.

Lose Condition

The user shocks the patient. Patient's cybernetic stability drops to 0.

2.3.4. Memory Solver

The player fixes the memory of the NPC by administering serums by clicking on the buttons that correspond to each serum. The player tries to administer these serums so that the three sine waves stay in a good range. When these sine waves stay in that range for a certain amount of time the player wins, if these sine waves go too far above or below the player loses (Figure 2.9).

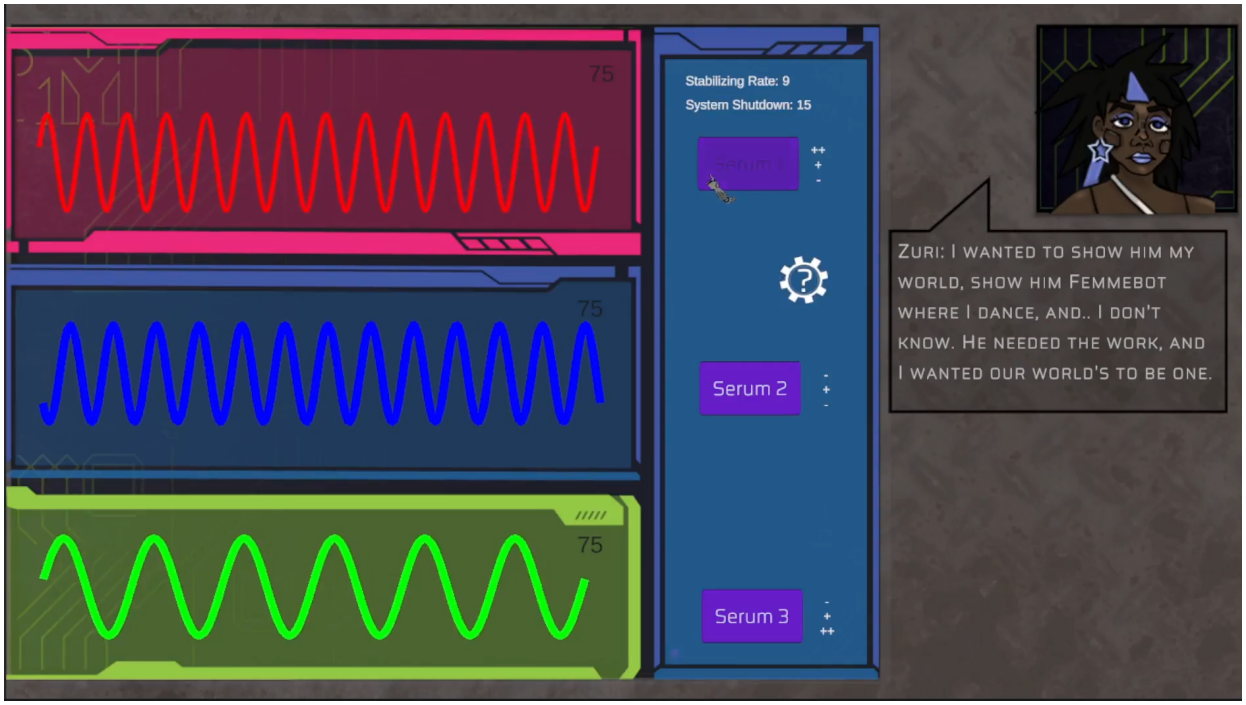


Figure 2.9

2.3.4.1. GamePlay

A. The user must apply the serum's to the patient in a way that stops all 3 sine waves from reaching 65 or below or 90 or above. Each serum adds to specific sine waves and subtracts from others. The effects of the serum's are next to their buttons.

2.3.4.2. Dialogue

A. Clicking a Dialogue option progresses the dialogue.
    a. Dialogue ends at some point or stops to wait for the player to finish part of the activity.

Win Condition

The virus is cleared from the brain implant and the patient survives.

Lose Condition

The virus overwhelms the brain implant and the player fails, resulting in either patient death or a day restart.

2.4. Script Functions

2.4.1. Script: ScrewReplace.cs

2.4.1.1. Description: This script is placed on empty objects with a box collider to detect if a screw enters the collider of the placement objects and then checks if it can be placed there.

2.4.2. Script: Screw.cs

2.4.2.1. Description: Script is genuinely for the movable object lets it know that the screws are. Pulls the information on the number of screws that are on the movable object and lets the object be moved or not moved and lets it know if it's moved from the original spot and back in the original spot.

2.4.3. Script: UpgradeObjectBehavior.cs

2.4.3.1. Description: This goes on the replaceable object, behind the movable object there is a replaceable object that acts in the same way as a screw in the fact that you can pick up and place it in the toolbar. Purpose: To make the object able to be picked up by the screwDriver.

2.4.4. Script: LaserLineRender.cs

2.4.4.1. Description: Technical copy of raycast.cs except you also render the line you raycast

2.4.5. Script: PlayerMovement

2.4.5.1. Description: Set speed, set groundrag, rigidbody

2.4.6. Script: MouseLook.cs

2.4.6.1. Description: Makes sure camera is in the right orientation as well as the rigid body

2.4.7. Script: MenuCameraController.cs

2.4.7.1. Description: Switching from the menu of the game to the actual game itself, more of a developer tool at the moment

2.4.8. Script: ScrewDriver.cs

2.4.8.1. Description: This is for picking up the screwdriver and placing any item that is a part of it, anything can be placed if it fits the constraints.

2.4.9. Script: Screw Mechanics.cs

2.4.9.1. Description: This is for allowing the screws to be picked up, moved, placed, and for the object to know that it's been removed.

2.4.10. Script: NewDialogueScript.cs

2.4.10.1. Description: It takes in the given dialogue script and loads the information and goes through the story.

2.4.11. Script: DialogueManager.cs

2.4.11.1. Description: Manages the dialogue system and controls the flow of dialogue

2.4.12. Script: DialogueTrigger.cs

2.4.12.1. Description: This script is used to trigger dialogue events when the player interacts with a specific object or NPC in the game.

2.4.13. Script: Drag3D.cs

2.4.13.1. Description: The script allows for objects to be moved in 3D space using the mouse.

2.4.14. Script: SettingsMenu.cs

2.4.14.1. Description: The script contains all the settings that the player can change in the game.

2.4.15. Script: ESCListen.cs

2.4.15.1. Description: The script listens for the Esc key to be pressed and freezes the time in game while it is pressed.

2.4.16. Script: DragHandler.cs

2.4.16.1. Description: The script handles the dragging of components within the repair broken devices activity by allowing those components to be picked up, dragged, and dropped all within a UI display.

2.4.17. Script: SnapController.cs

2.4.17.1. Description: This script handles the snapping of components after they are dropped by the user within the repair broken devices activity UI. If the object is close to a snappable location it will snap that object to the location. It also checks and handles most of the game's win/loss conditions based on snapping.

2.4.18. Script: LineMeter.cs

2.4.18.1. Description: Calculates the amount left based on how much has been drawn

2.4.19. Script: PaintColor.cs

2.4.19.1. Description: Multiple functions to add or subtract colors

2.4.20. Script: DrawLine.cs

2.4.20.1. Description: Draws a line within the spraypaint activity. It follows the mouse as the user is holding left-click.

2.4.21. Script: GameManager.cs

2.4.21.1. Purpose: Manage the game with a state machine.

2.4.22. Script: Interactable.cs

2.4.22.1. Description: Abstract class for Interactives

2.4.23. Script: CameraTrigger.cs

2.4.24. Script: OpenSignController.cs

2.4.25. Script: Raycast.cs

2.4.25.1. Description: Sends out a raycast and checks if it's an interactable

2.5. Unity

2.5.1. Version 2022.1.17f

2.5.2. HDRP (High Definition Render Pipeline)

2.6. User Interface

2.6.1. Main Menu

2.6.1.1. Continue button - starts the game from the last saved point.

2.6.1.2. Start button - starts the game from the beginning

2.6.1.3. Options button - leads into a submenu with the following options:

A. Video - menu with adjustments to any visual elements.

B. Audio - menu with adjustments to any audio and narrative elements.

2.6.1.4. Credits - credits display of project members and their respective roles.

2.6.1.5. Exit Button - Exits the game.

2.6.2. Pause Menu

2.6.2.1. Continue button - unpauses/continues the game.

2.6.2.2. Restart Day button - restarts the current day.

2.6.2.3. Restart Game button - restarts the full game.

2.6.2.4. Options button - opens the Options menu.

A. Video button - leads into a submenu with the following options:

B. Video - menu with adjustments to any visual elements.

C. Audio - menu with adjustments to any audio and narrative elements.

2.6.2.5. Credits - credits display of project members and their respective roles.

2.6.2.6. Exit to Menu button - exits to Main menu.

2.6.3. In-Game UI

2.6.3.1. Activity UIs

- Feature dialogue boxes on the side along with the current NPC and their mood.

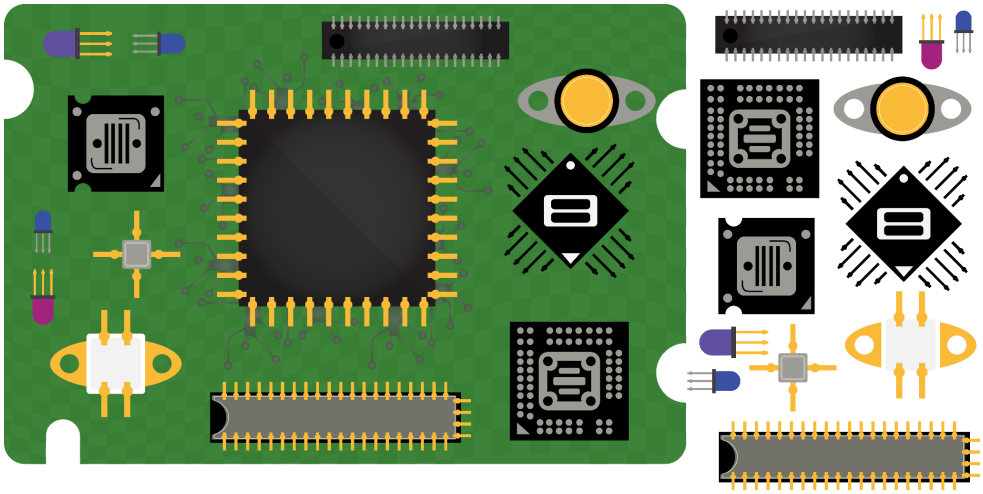


Figure 2.8: Repair Broken Devices UI

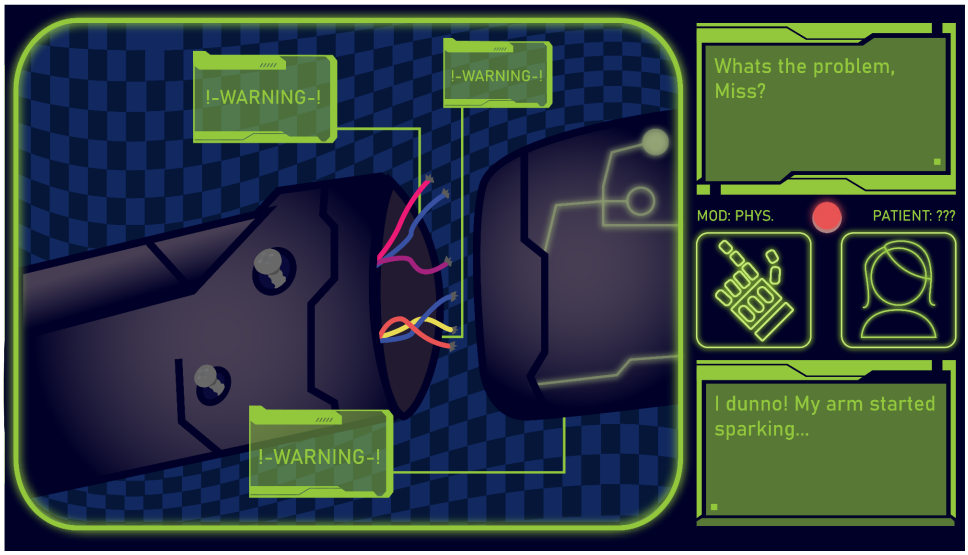


Figure 2.9: Concept art for Activity UI

2.7. Audio

2.7.1. Background Music (BGM)

2.7.1.1. The game adjusts the BGM based on state in the game given states in Wwise and in Unity.

2.7.2. Sound Effects (SFX)

2.7.2.1. The game has SFX for the player, their interactions with the environment, and for elements in the activities.

3. Non-functional Requirements

3.1. Human Factors

3.1.1. Types of Users

3.1.1.1. Users expect to be playing a game and have a device on which they can play it.

3.1.1.2. Playtesters who know the workings of the game and understand where there might be issues such as bugs.

3.2. Documentation

3.2.1. In addition to this document, a GDD is available with an in-depth explanation of the game's mechanics, characters, story, and gameplay elements.

3.2.2. Scripts are outlined in a standalone document which explains everything the script does in-depth.

3.2.3. ClickUp and Perforce provide a history and plan for the game’s development.

3.3. Hardware

3.3.1. The client must have the following hardware:

3.3.1.1. A computer

3.3.1.1.a. OS: Windows 7 or higher

3.3.1.1.b. Processor: 2Ghz or higher

3.3.1.1.c. Memory: 4GB Ram

3.3.1.1.d. Graphics: Geforce 750ti or higher

3.3.1.2. A keyboard

3.3.1.3. A mouse/pointing device

3.3.1.4. A monitor

3.4. Performance

3.4.1. Response Time

3.4.1.1. No trailing of base systems should be expected on lower-end computers. As such users should not expect objects to lag behind their mouse in use and system slow-downs.

3.5. Error Handling and Reliability

3.5.1. Unity Built-In Test Framework

3.5.1.1. Utilizes built-in framework to provide wide range code coverage

3.5.1.2. Allows for both integrated, Unit, and performance testing

3.6. Security

3.6.1. N/A

3.7. Code Requirements

3.7.1. Focus on legible code.

3.7.2. Comment sections on their general usage.

3.7.3. Make understandable variables and functions.
3.7.4. Follow a standardized caps usage convention, such as camelcase for variables.
3.7.5. Decouple most of the code.
3.7.6. Reduce spaghetti code via functioning out areas that can be their own functions.
3.7.7. Don’t overuse one script for multiple functions.
3.7.8. One script, one function.

4. System Evolution

System Evolution

1. Development operates using the Unity game engine, with regular updates and upgrades to ensure compatibility with the latest version of the engine.
2. Perforce version control system manages the game's codebase, with regular backups and versioning to ensure ease of collaboration and rollback in case of issues.
3. AWS hosts and stores the game's files and assets, with regular backups and scalability to ensure ease of access and reliability.
4. The game is tested on a variety of devices and platforms to ensure compatibility and performance.
5. The game has regular updates and patches to fix bugs and add new features, based on feedback from players and the development team.
6. The game has a system in place to track and analyze player usage and behavior, to inform future updates and improvements to the game.
7. The game has a system in place to allow for easy integration of new content and assets, to ensure ease of future expansion and customization.
8. "Cyber Surgery": Players take on the role of a cyberpunk doctor performing advanced cybernetic surgery on patients. They must use precision tools and techniques to complete the surgery while avoiding complications.
9. "Hackers Heart": Players must navigate a virtual network to find and fix a virus that is attacking a patient's implanted cybernetic heart. They must use hacking skills and knowledge to find and remove the virus before time runs out.
10. "Robo-Reboot": Players take on the role of a cyberpunk doctor working to repair a malfunctioning robot. They must use diagnostic tools and problem-solving skills to find and fix the issues, while also avoiding causing further damage.

11. "Neural Net": Players take on the role of a cyberpunk doctor working to repair a patient's neural implants. They must use precision tools and knowledge of neural networks to repair the damaged connections, while also avoiding causing further damage.
12. "Cyber Crime Scene": Players must investigate a cybercrime scene in which a patient has tampered with cybernetic implants. They must use forensic skills and knowledge of cybercrime to find and arrest the perpetrator before they strike again.