THE MEMORY CHAMBER

A Psychological Thriller Game Design Document

4th Milestone

Jasmina Vulović, Aleksandar Arnaudov, Louis Sajuk, Muhammed Bayram

January 2026

Chapter 1

Game Description

Introduction and Game Overview

The Memory Chamber is a First-Person puzzle game centered on the course theme of Recursion and Time Travel. The protagonist, **Dr. Elias Thorne (The Amnesiac)**, is trapped within a self-perpetuating 30-minute time loop inside a single, isolated research bunker. The loop resets every 30 minutes, triggered by a sudden, violent act: the murder of his partner, **Dr. Alice Vance (The Companion).** The player's journey is a terrifying descent into self-discovery as they realize the recursive killer is none other than a persona within themselves.

The game's design forces the player to solve a puzzle based on fragmented memory. Elias is the only entity aware of the time jumps (via the persistent notes), trapping him in a unique psychological isolation. Alice, the victim, is completely unaware of the looping horror. The core technical achievement—the State-Driven Memory Wall System—visually embodies this recursive trauma, where essential, state-triggered memories are recorded and then dynamically corrupted over repeated cycles.

Narrative Backstory and Setting

The Setting: The Silent Bunker

The entire experience takes place within a confined, two-room, Cold War-era Research Bunker—The Chamber. The aesthetic is a crucial element of the psychological horror: Late 1970s Analog Horror. The environment is deliberately small, featuring heavy concrete walls, buzzing fluorescent lights, bolted metal, and antiquated technology (flickering CRT monitors, antiquated radio equipment). This oppressive setting justifies the complete lack of external interference, forcing the focus onto the internal, recursive conflict.

The Chamber consists of a Main Habitation Area and a Utility/Control Room, which houses the Time Jump Device console and the crucial Amnesia-Resistant Plating, the writing surface for the persistent memory system.

The Background: The Vance-Thorne Research

Dr. Elias Thorne and Dr. Alice Vance, his partner and colleague, were engaged in highly sensitive research into Time Jump Stabilization. While they succeeded in creating a stable, localized 30-minute reverse time jump, the breakthrough came with a catastrophic side effect: Cognitive Destabilization. The brain could not reconcile the instantaneous rewriting of its past, leading to profound psychological fracturing.

To mitigate this, they developed the Memory Observation Unit (MOU), a system of specialized, plasma-coated metallic patches (The Plating) installed on the walls. The MOU was designed to passively absorb and stabilize the user's critical thoughts during a jump, allowing memory fragments to persist.

The Storyline: The Fracture and The Trigger

The Scientific Trauma and Persona Genesis: The immense stress, coupled with suppressed guilt from the experiment's ethical lines, caused a catastrophic personality split in Elias. The side effects of the successful time jump enabled the creation of a violent, unconscious alter-ego—the Killer Persona—the physical manifestation of Elias's trauma and self-loathing.

The Opening Scene and Loop Trigger: The loop begins with a traumatic transition. The Amnesiac (Elias) violently snaps back to consciousness, disoriented and perhaps still holding the weapon. He finds Alice severely wounded but still briefly conscious. Alice succumbs to her injuries within the first minute, her death providing a final, traumatic accusation: a desperate, cryptic clue whispered before her last breath.

The Recursive Cycle: Overwhelmed by this traumatic loss and driven by the subconscious desire to prevent the death he just witnessed, the Amnesiac's mind immediately forces the Time Jump Device's emergency reverse-jump function, sending his consciousness back 30 minutes. Since the murder is committed by the Killer Persona at the 29:59 mark, the guilt-ridden Amnesiac is perpetually doomed to trigger the reverse jump, creating a self-perpetuating, scientifically justified recursive cycle. Alice and the environment are reset and completely unaware of the time jumps, isolating Elias in his repetitive horror.

Gameplay and Core Loop Mechanics

The game is a cycle of Investigation, Accumulation, and Psychological Breakdown, forcing the player toward a non-physical solution.

The 30-Minute Cycle

The loop is strictly governed by the timer:

1. Waking and Discovery (0:00 – 0:30): The Amnesiac wakes. The player's first action is always to rush to the Memory Wall to assess the new landscape of inherited thoughts. The presence of the notes is the only confirmation of the loop.

2. Investigation and State-Triggering (0:30 – 25:00): The player uses the notes to guide their actions. The core loop now revolves around executing specific, complex action chains to trigger the appearance of new, vital memories (notes).
   - Example: Only by correctly setting the frequency on the antiquated radio (Action X) will Elias subconsciously "remember" the pre-defined note: "The music is the trigger. Loop 14." (Note Y). If Action X is not performed, Note Y will never appear. This transforms the exploration into a state-driven search for memory fragments.

3. Interaction with Alice: Alice is the player's emotional thermometer. Her dialogue shifts from confused concern to outright fear and desperate defense as the player's actions, guided by the increasingly erratic notes, become more paranoid and aggressive. Her reactions are a crucial layer of feedback for the player's psychological state.

4. The Crisis (25:00 – 30:00): Environmental anomalies increase—lights flicker, static screams from the monitors, and a rapid, aggressive heartbeat audio cue pulses. This signals the increasing dominance of the Killer Persona.

5. The Reset (30:00): The final, suppressed murder occurs, and the time jump is triggered, repeating the cycle. All physical objects and Alice's state are instantly reset.

The Madness Meter: The Recursive Corruption

The Amnesiac's increasing knowledge of the loop, combined with the trauma of repeating the murder, causes their psyche to degrade. This trauma is reflected in the visual corruption of the persistent Memory Wall.

- **Progressive Corruption:** As the Madness Score (MS) (which increments every loop) rises, the notes—regardless of when they were triggered—become visually unreliable. Clean, early-loop notes become smeared with red, blurred, and are obscured by non-textual graphic noise.
- **The Breakdown:** Later loops reveal the tragic reality: the player's own most crucial memories (the notes) are now failing them, becoming a torrent of conflicting instructions and terrifying accusations (e.g., "DESTROY TAPE" written over "THE TAPE IS THE ANSWER"). This recursive failure of the memory system is the ultimate psychological horror.

Core Game Mechanics and Systems

The Memory Chamber targets a mature, narrative-driven audience. The overall gameplay is a high-pressure cycle of Investigation, Accumulation, and Breakdown within a 30-minute time limit. The primary objective is to break the recursive loop by finding a non-physical, psychological solution to the protagonist's self-destructive persona.

The gameplay is built upon two intertwined core systems:

The State-Driven Memory System

This is the central puzzle mechanic. Due to the protagonist's amnesia, memories are stored as persistent notes on the Memory Wall. Instead of player input, these notes are pre-defined memory fragments tied to specific in-game states. Only by correctly performing a complex action (e.g., tuning the radio to a specific frequency) will the associated memory flag be set to TRUE. At the end of the loop, the system renders all discovered notes onto the persistent Memory Wall Texture (MWT), creating a cumulative, self-referential record for the next cycle. This system requires robust state machine design and external texture caching to achieve recursive data persistence.

The Madness System

This system tracks the protagonist's escalating trauma. The Madness Score (MS) is an integer that increases with every failed loop, directly translating the recursive stress into sensory deterioration. The MS is fed into a custom Fragment Shader applied to the MWT, causing the persistent notes to become increasingly visually corrupted—blurring, bleeding, and shimmering—representing the decreasing sanity. High MS also triggers environmental auditory and visual anomalies (e.g., loud heartbeat sounds, screen distortion) that are only perceived by the player, effectively making the game environment the physical manifestation of the protagonist's psychological breakdown. The solution to the loop is only attainable when the player learns to decipher the truth hidden beneath the madness.

The End Game: Breaking the Recursive Cycle

The solution to the puzzle is the final, critical State Trigger—the identification of the core psychological intervention required to neutralize the Killer Persona.

The Final Action: The accumulated (and corrupted) notes eventually point the player to a sequence of actions that leads to the final, non-physical trigger (e.g., smashing the mirror where the violent reflection appears; neutralizing the audio trigger by destroying the tape).

The Psychological Intervention: In the final, successful loop, the Amnesiac executes the corrective action, and the violent impulse is subdued.

The Resolution: At 30:00, the lights flicker, the impulse hits, but the murder fails to execute. The Time Jump Device is not activated. The game ends at 30:01 with the Amnesiac and Alice alive, but traumatized. The scientific recursion is broken, but the Amnesiac must now consciously live with the memory and guilt of the trauma he was suppressing—a final, chilling psychological ending.

Chapter 2

Technical Achievement

Our game focuses on the recursive aspect of trying to use a time machine to save your partner. Multiple key mechanics emphasize the toll of continually going back in time. These indicate the player's progress towards the end of the game. With each try, the player will sense a change in the environment and perception, which grows darker and darker with each loop. Additionally, the player can also find notes from different timelines, or in this case, from other players.

Incremental Loop Manager

To realize the growing toll of time-traveling on the user's mind and body, each loop will add effects of a diminishing mental state. A manager element will take control of all changes implemented within each loop. Certain key effects include:

- Notes the player can read will become messier
- The player can hallucinate
- Lighting color and intensity may adapt
- Applying maps/patterns to objects for visual deterioration that evolves each loop

Server-side Communication for Player Notes

A key feature allowing the player to find clues for the current loop's objective is finding and collecting notes. Through the nature of time travel, the player is able to find notes that they have not written themselves physically. Other players can write notes and cues, which can then be found by other players. To prevent users from spoiling others, notes can only be found in the same loop where they are written and can only contain a set of predetermined words. The back-end configuration will contain:

- Communication system between the game and a server to store user notes
- Notes can be stored online together with additional data (current loop)
- Ability to access the correct set of user notes and traverse information into the game

Both technical achievements closely support the actions of repeated time travel. They are also intertwined as they depend on the current game state and player progress.

Chapter 3

Big Ideas Bullseye

3.1 Core Idea

The core idea behind our game is to explore the psychological and temporal consequences of a failed time experiment. The player takes the role of a scientist trapped within a self-perpetuating 30-minute time loop inside a research bunker, following a catastrophic laboratory accident. Each reset begins with the death of the companion, whose murder serves as both the loop's trigger and its unresolved mystery. Every cycle reveals subtle differences in the environment: objects shift, notes remain, and the companion's behavior grows increasingly unpredictable as the player searches for a way to break the loop and uncover the truth.

While the physical world resets after each jump, the Memory Wall retains the notes recorded during previous loops, serving as the only persistent record of the protagonist's fragmented consciousness. These notes, automatically generated by the state of the world, act as memory fragments rather than deliberate writings, capturing subconscious thoughts and critical discoveries. To progress, the player must carefully perform specific actions to unlock these memories, connecting them across cycles to reconstruct the events that led to the experiment's failure and the companion's death. With each new loop, the accumulated knowledge exposes both hidden truths and growing instability within the protagonist's fractured mind.

A strict 30-minute time limit adds constant pressure, forcing the player to decide what memories to pursue before the cycle resets. As the Madness System progressively distorts the once-stable Memory Wall, the challenge becomes more than solving a mystery; it becomes a race against psychological collapse. The idea is to challenge the player to navigate both temporal and mental recursion, piecing together a shattered identity to prevent the inevitable tragedy at the heart of the experiment.

3.2 Technical Innovation

Our main technical innovation is a **Game State Manager** that dynamically links the player's psychological state and persistent data across recursive time loops. This manager oversees two intertwined systems that evolve together as the game progresses: a **server-based note persistence system** and a **madness progression mechanic**.

The first component enables server communication for player-written notes. Each note created by the player can be uploaded and retrieved from an online database, allowing the world to be influenced by fragments left by other players in similar loops. To prevent users from spoiling others, notes can only be found in the same loop where they are written and can only contain a set of predetermined words. These shared notes act as "echoes" from parallel timelines, reinforcing the sense of collective memory and isolation.

The second component, the **Madness System**, tracks the protagonist's escalating psychological in-

stability caused by repeated time jumps. The Madness Score increases with every failed loop, visually reflected in the environment and UI such as distorted handwriting, blurred text, and static interference. As the Madness Score rises, these effects expand to environmental elements including lighting, sound, and visual distortion, gradually transforming the entire world into a manifestation of the protagonist's mental breakdown. This system directly interacts with the Game State Manager, ensuring that both the environment and the companion's behavior dynamically adapt to the player's current state of mind.

Together, these two systems create a seamless technical and narrative bridge between memory, recursion, and psychological decay, turning the game state into a living reflection of the player's fractured identity and persistence.

3.3 Stretch Layer (Future Potential)

The game could evolve by introducing adaptive puzzle logic that changes based on the player's previous performance. Clues and their meanings could be influenced by the notes the player wrote in earlier loops, allowing each cycle to feel unique and reactive. Different time limits could also be introduced to adjust tension, with shorter loops increasing urgency and variable-length cycles simulating the instability of memory. These systems would create a more dynamic and unpredictable experience while preserving the core recursive structure of the game.

Chapter 4

Development Schedule

4.1 Targets

Functional Minimum

- FPS Controller (movement, looking, interaction (inspect/pick up objects/drop objects))
- Base environment (lab/break room) with static props
- Simple companion model with idle animation (including walking on static waypoints)

Low Target

- Static note system (player can see pre-posted notes the developers made)
- Room with a "memory wall"
- Refined base environment
- Trigger event that starts the first time loop
- Basic Start/Exit/Pause Screens
- Loop mechanism (state of the game is saved based on the player's progress)

Desirable Target

- Amnesia/Sanity system (some basic negative effects)
- Basic sanity system(simple effect on player and companion)
- Companion Logic (not idle, follows around, reacts to certain actions)
- Basic Win/Lose conditions
- Basic Audio: background music, step music, pick up/drop sounds/ some audio for the time travel
- Basic UI
- Game loop: Find password
- Game loop: Find the murder weapon
- Game loop: Prevent murder (hide weapon/destroy/tell her to hide)

High Target

- Audio polish: Improved puzzles with some audio clues/not only static
- Online note system improved: We have logic to control what notes are allowed to be on the memory wall and the player can get a certain amount of notes; players can rank notes
- Improved amnesia system
- Refined sanity system (elaborate visual effects and more complex companion reaction to player's sanity decrease)

Extras (Post-Semester)

- Additional rooms.
- More objectives.
- Longer gameplay.

4.2 Core Idea

Task Name	Assignee	Deadline	Planned	Actual
Docs: Brainstorm on game ideas	All	29.10.2025	4	6
Docs: Writing the project report	All	29.10.2025	4	8
Docs: Draw out sketches	All	29.10.2025	2	2
Docs: Prepare the presentation	All	29.10.2025	2	2

4.3 Game Prototype

Task Name	Assignee	Deadline	Planned
Create Physical Prototype	All	12.11.2025	6

4.4 Interim Demo

Task Name	Assignee	Deadline	Planned
Create Git Repo, Project in Unity	Louis	03.12.2025	1
Set Up Unity Project (Version Control, Scene Setup)	Louis	03.12.2025	1
UI: Search/Design for the Assets (overall aesthetic of the game)	All	03.12.2025	4
Player: Basic Player Object Movement	Muhammed	03.12.2025	3
Player: Sanity System Skeleton	Aleksandar	03.12.2025	2
Player: Object pick up/drop	Muhammed	03.12.2025	2
Companion: Simple design	Aleksandar	03.12.2025	8
Level Design 1: Initial room	Jasmina	03.12.2025	8
Level Design 2: Memory Wall	Louis	03.12.2025	8
Static note system	Muhammed	03.12.2025	8
Time travel system	Jasmina	03.12.2025	8
Simple game loop	Aleksandar	03.12.2025	1
Cut scene	Aleksandar	03.12.2025	4

4.5 Alpha Release

Task Name	Assignee	Deadline	Planned
UI: Sanity Bar	Muhammed	07.01.2026	3
Online Note System	Muhammed	07.01.2026	12
Sanity System (Basic side effects)	Aleksandar	07.01.2026	5
Refined Companion Logic	Louis	07.01.2026	8
Basic Win/Lose conditions	Louis	07.01.2026	3
Basic UI	Jasmina	07.01.2026	4
Game Loop: Find the password	Aleksandar	07.01.2026	6
Game Loop: Find the murder weapon	Louis	07.01.2026	6
Game loop: Prevent the murder	Muhammed	07.01.2026	6
Audio: Search/Create assets	Louis	07.01.2026	3
Shaders: Create/ Search	All	07.01.2026	8

4.6 Playtesting

Task Name	Assignee	Deadline	Planned
UI: Action Prompts	Aleksandar	21.01.2026	10
UI: Narrative Text	Jasmina	21.01.2026	8
UI: Narrative Clues	All	21.01.2026	20
UI: Menu	Jasmina	21.01.2026	3
UI: Start Screen, Pause Screen	Louis	21.01.2026	4
Improved Online System	Muhammed	21.01.2026	5
Improved Sanity System	All	21.01.2026	20
Audio Polish	All	21.01.2026	20
Playtesting	All	21.01.2026	2
Bug Fix	All	21.01.2026	yes

4.7 Final Release

Task Name	Assignee	Deadline	Planned
Polishing	All	04.02.2026	tbd
Bug Fix: final	All	04.02.2026	tbd
Docs: Final Documentation	All	04.02.2026	10
Docs: Game video	All	04.02.2026	1
Docs: Slides	All	04.02.2026	2

Chapter 5

Assessment

Why is our game cool?

- Psychological Time-Loop Tension: Each loop resets the world, but not the player. The environment becomes darker, notes decay, the companion behaves differently, and the character slowly loses sanity. The more the player retries, the more unstable reality becomes.
- Persistent Notes Across Timelines: Players leave notes that survive each time jump and can also discover notes written by other players in parallel loops. These clues feel like whispers from alternate timelines, reinforcing mystery, cooperation, and uncertainty.
- A Mystery Solved Through Memory, Not Combat: The player pieces together clues, reads messages, interprets changes, and eventually prevents the companion's death. The tension comes from limited time, repeated failure, and the character's degrading mental state.

Who might want to play this game?
The game appeals to players who enjoy narrative mysteries, psychological thrillers, subtle horror, and environmental storytelling. Fans of games such as *Outer Wilds*, *SOMA*, *PT*, and *Observer* will be drawn to uncovering clues and solving the overarching mystery.

What do players do?
Players explore a sealed research facility, collect information, read notes, observe differences between loops, and uncover how to prevent the murder. Each loop reveals new details, but also pushes the character deeper into psychological instability.

What world does the system simulate?
The world is a closed laboratory trapped in a recursive time loop. Most objects reset between loops, while written notes and the player's sanity persist. With every restart, the environment becomes more distorted and less reliable, reflecting the mental toll of repeated time travel.

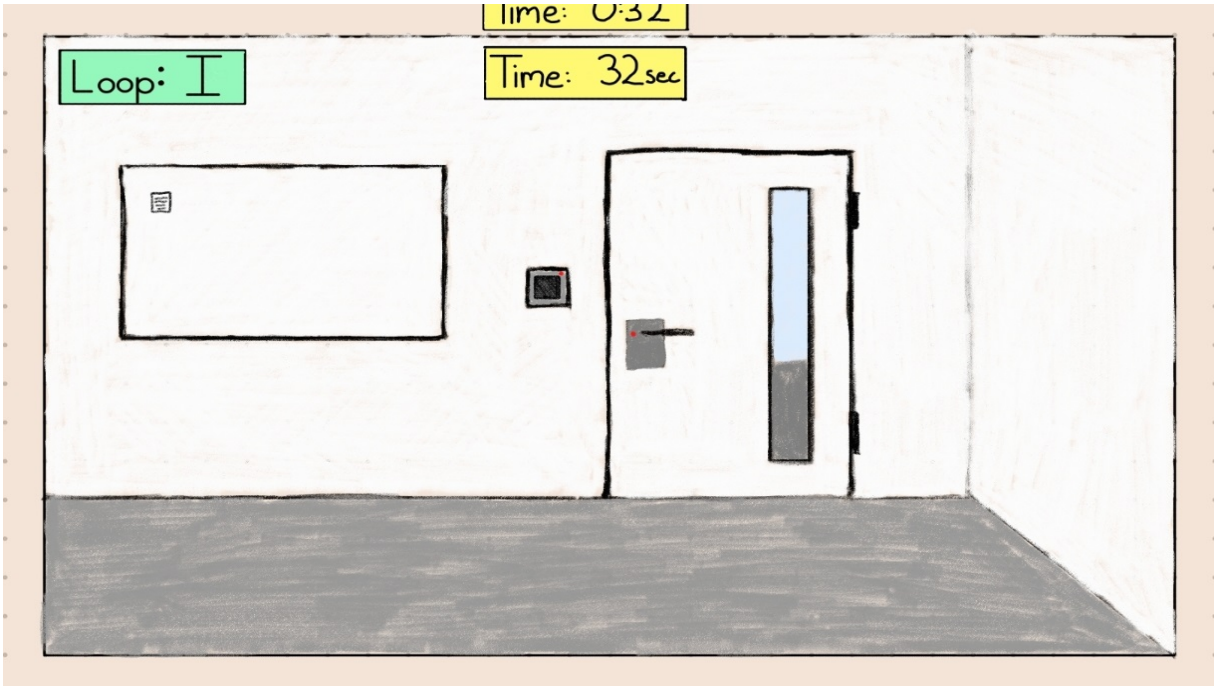
What makes the game a success? The design will be successful if:

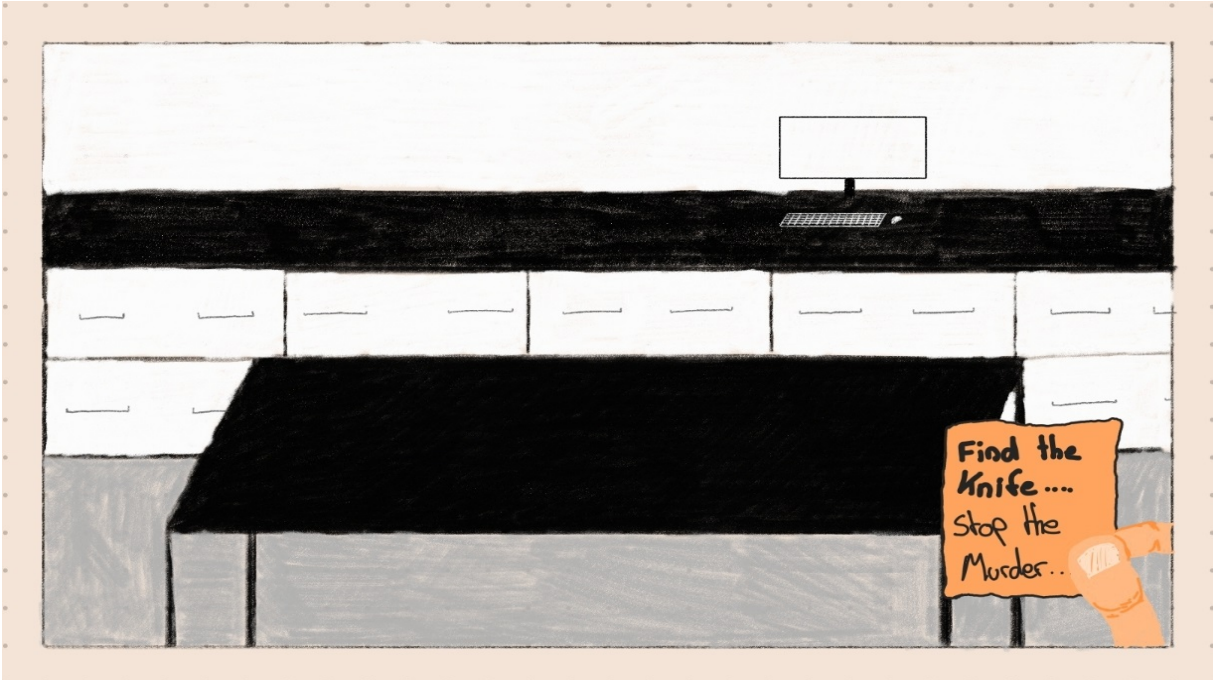
- Each loop feels progressively different and more unsettling.
- Persistent notes feel meaningful and help players advance.
- The sanity system creates pressure and emotional tension.
- Players feel motivated to try again after each restart.
- The final goal of saving the companion feels earned through deduction.

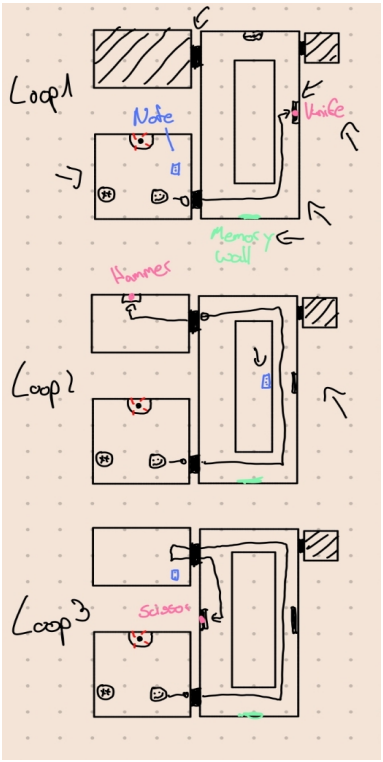
If players finish a loop thinking, "I know more now – I need to try again," then the core vision of the game is achieved.

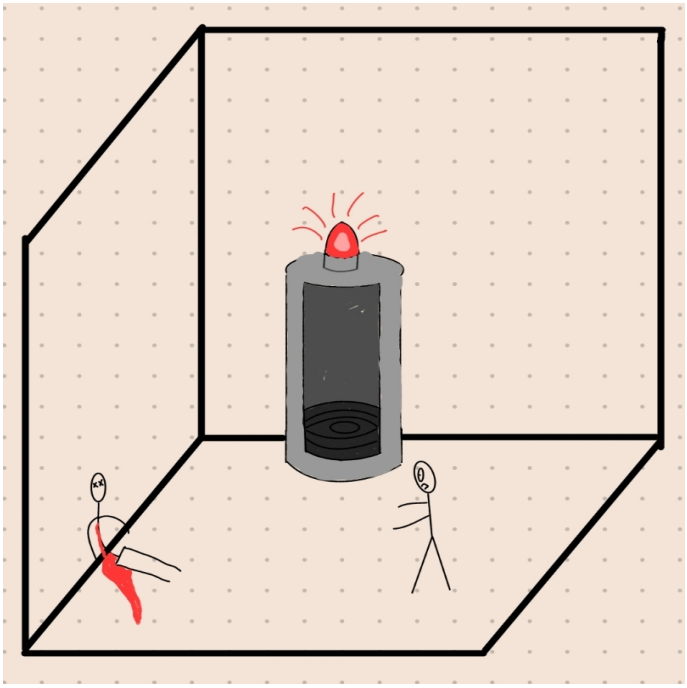
Chapter 6

Sketches



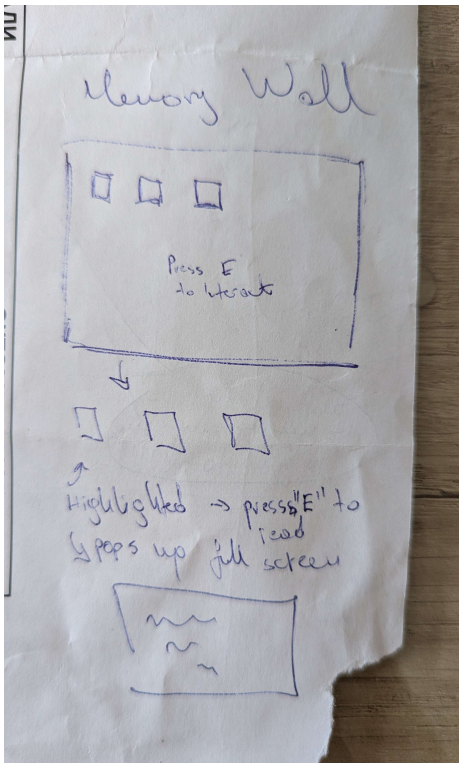



Loop1
Note
Knife
Memory Wall
Hammer
Loop2
Loop3
Scissors

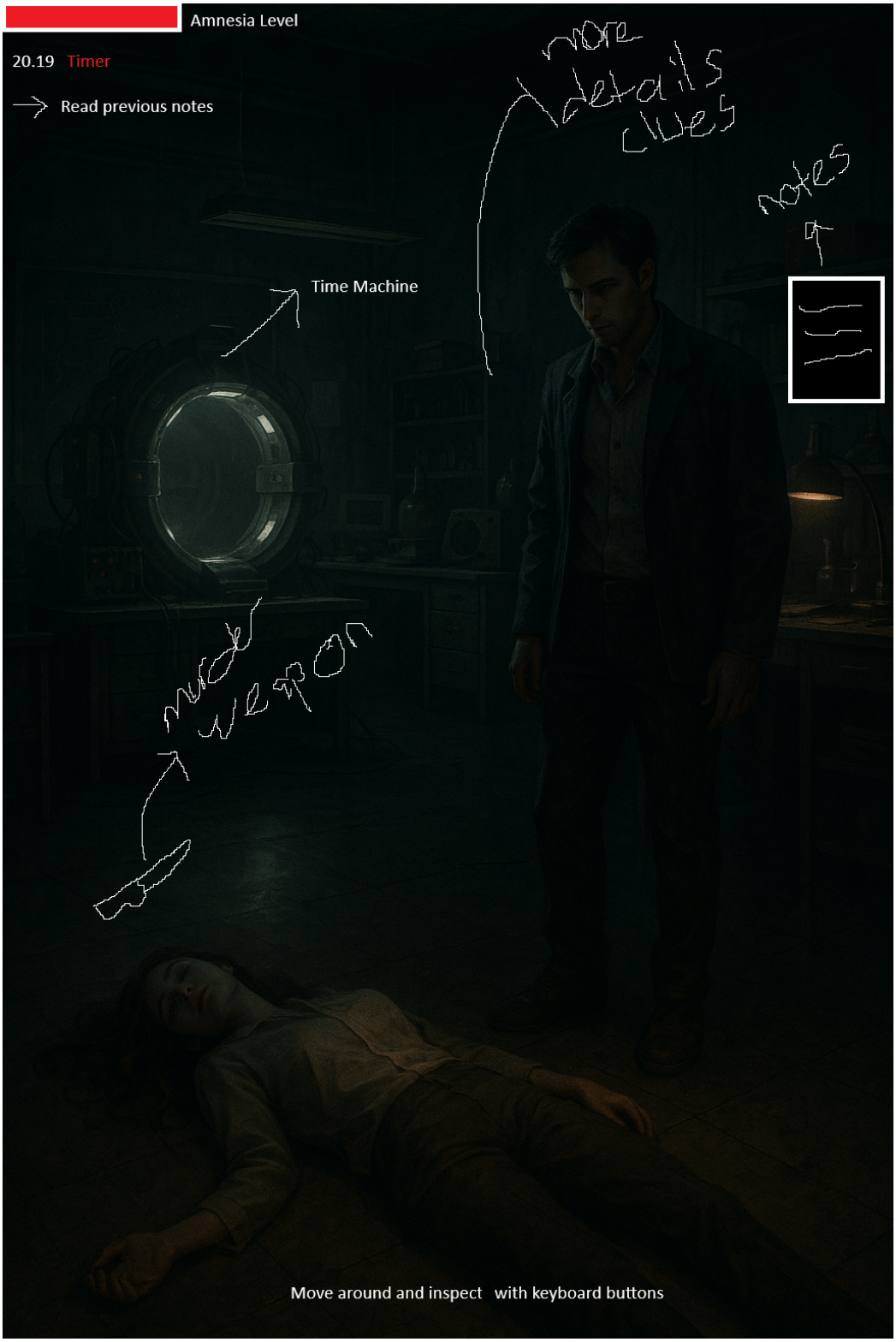


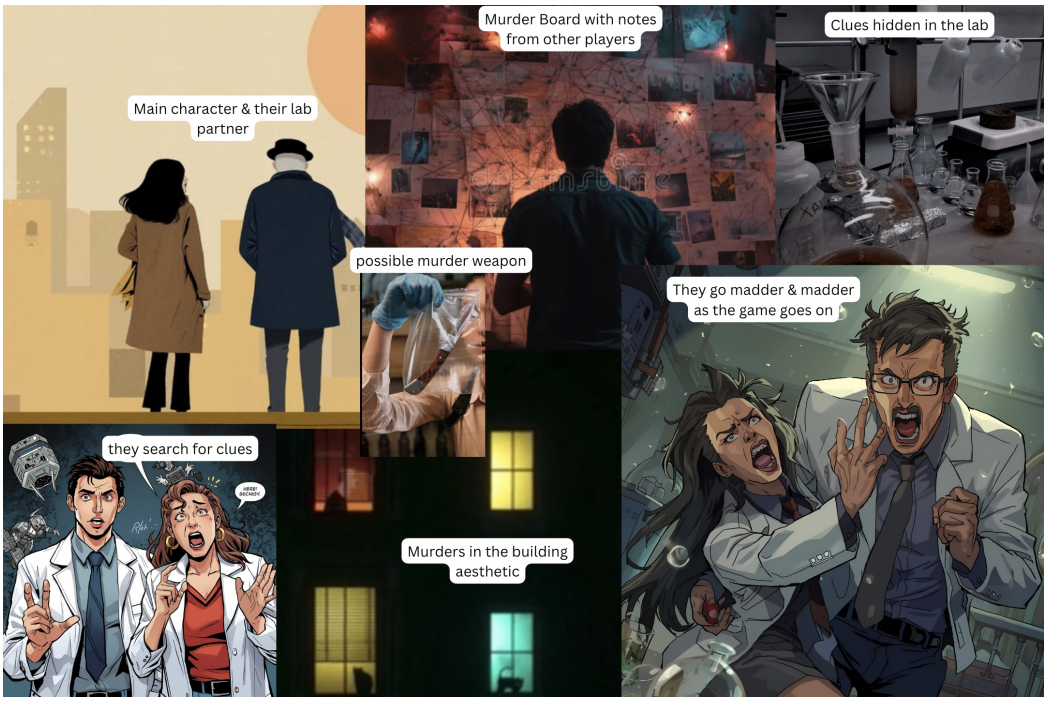

GO BACK
Time machine Room?


Break Room?
Wait, hold up, you're not making any sense I'm going to die!?!?


Memory Wall
Press E
to interact
Highlighted → press "E" to
read
pops up full screen


GO BACK
Time
machine
Room
?


Amnesia Level
20.19 Timer
Read previous notes
more details clues
notes
Time Machine
Move around and inspect with keyboard buttons


Main character & their lab partner
Murder Board with notes from other players
Clues hidden in the lab
possible murder weapon
They go madder & madder as the game goes on
they search for clues
Murders in the building aesthetic

Chapter 7

Prototype

Prototype Design and Construction

To test the core gameplay of The Memory Chamber, we created a physical paper prototype simulating the two-room laboratory where the player experiences the time loop. The prototype includes interactable puzzle elements, a memory recording system, and indicators for sanity and time effectively modeling the recursive structure of the digital version.

The layout features:

- A door PIN keypad puzzle connected to scientific clues.
- A Memory Wall, represented with sticky notes, where players write discoveries that persist across loops.
- A timer (30:00) and a sanity level indicator, visualizing the tension and psychological decay across cycles.
- Player and Companion markers (P / C) showing their positions during the gameplay.
- Illustrated puzzle objects (scientific tools, notes, equations) that guide the player toward discovering the cause of the murder and the loop trigger.

Gameplay Experience

The prototype successfully simulated the core gameplay of our game, focusing on the essential mechanics rather than visual or technical polish. This approach allowed us to isolate and test the fundamental elements of the experience without being distracted by secondary details. Following the goal of this milestone, we used the prototype to better understand what truly defines our gameplay and how each mechanic contributes to the overall tension and pacing of the loop.

- The player investigates the lab and interacts with objects to uncover clues.
- Puzzle solutions (e.g., solving wavelength equations) reveal PIN numbers that unlock the next room.
- After solved puzzles, the player records the discovery on the Memory Wall.
- At the end of the loop, the timer resets, and the player starts again, but the Memory Wall notes persist, guiding the next attempt.

Was it fun?

Yes, the group found the experience enjoyable even though it was quite challenging to create. It was fun to be able to play the game we had only imagined so far, even without having a digital version yet.

What we have learned?

We did not expect the prototype to be this useful, but once we completed it and started playing, we noticed several design details and potential issues we had not considered before. It turned out to be more challenging than we anticipated to connect the story in a consistent way. We were not fully aware of this issue in earlier stages, but testing the prototype made it clear. We also realized that our puzzle system requires further development and refinement. This process allowed us to test our core gameplay mechanics in practice and better understand how our main systems interact. It helped us refine the overall structure of the game and identify which elements worked effectively and which needed improvement. Based on these findings, we made several design adjustments summarized below.

Changes Added

- The player now begins in the second room, discovering the dead companion at the start of the game.
- Introduced a murder scene recreation mechanic before the first time travel to establish narrative context.
- Defined clear objectives and interactions for both the first and second rooms.
- Expanded the range and logic of puzzle ideas to improve gameplay variety.
- Fixed several plot inconsistencies identified during playtesting.
- Added a conclusive trigger concept (time machine has a blackout)
- Fixed several plot inconsistencies identified during playtesting.

Sketches:

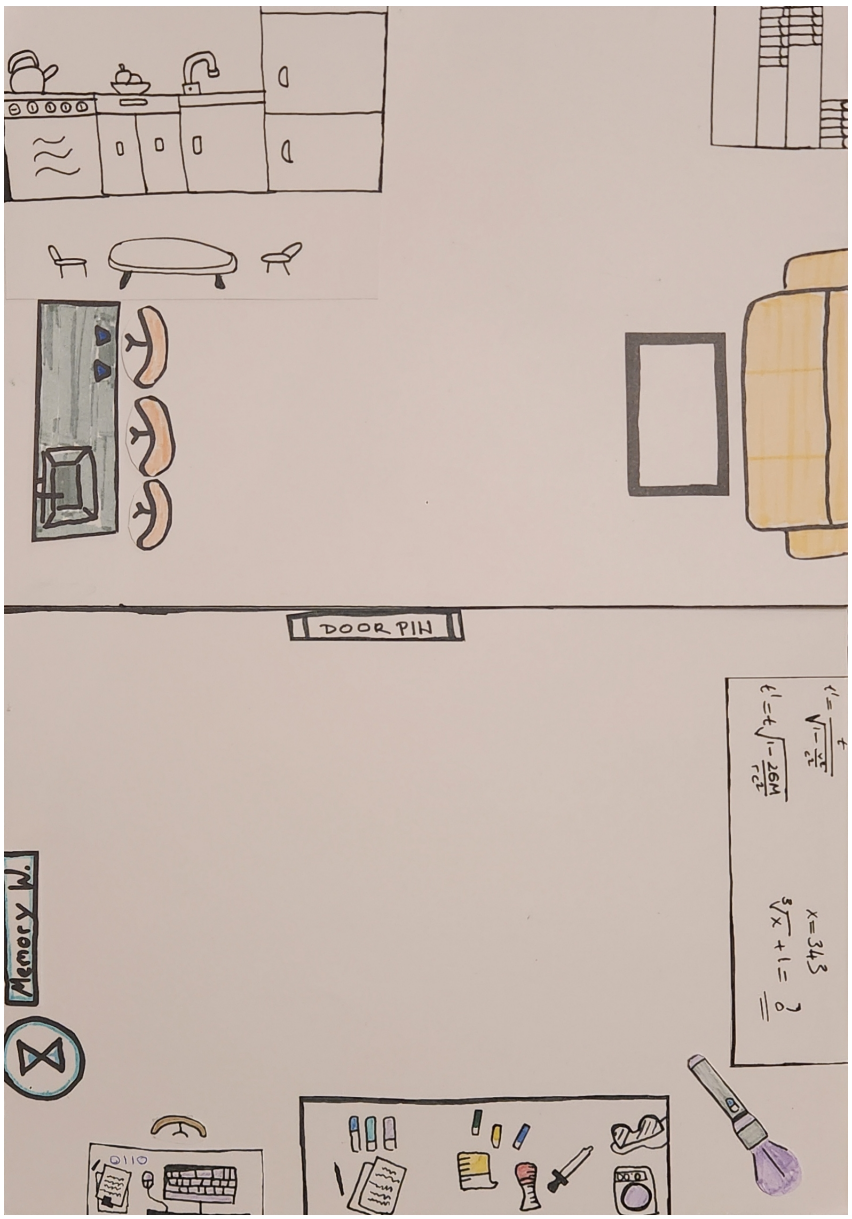


Figure 7.1: The full 2-room overhead layout

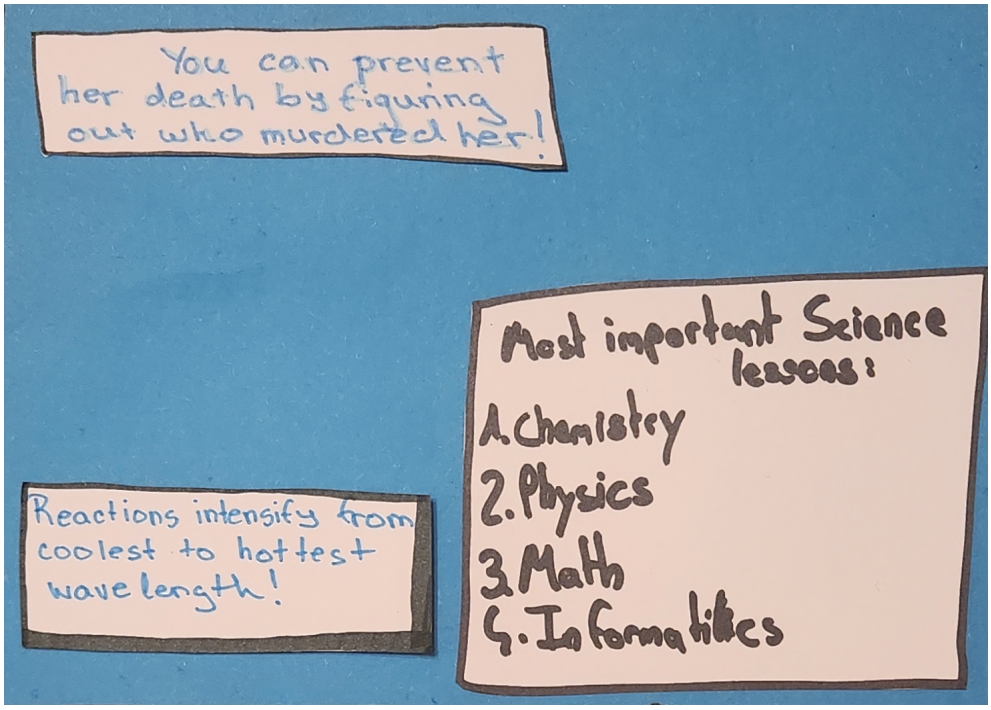


Figure 7.2: Memory wall screen

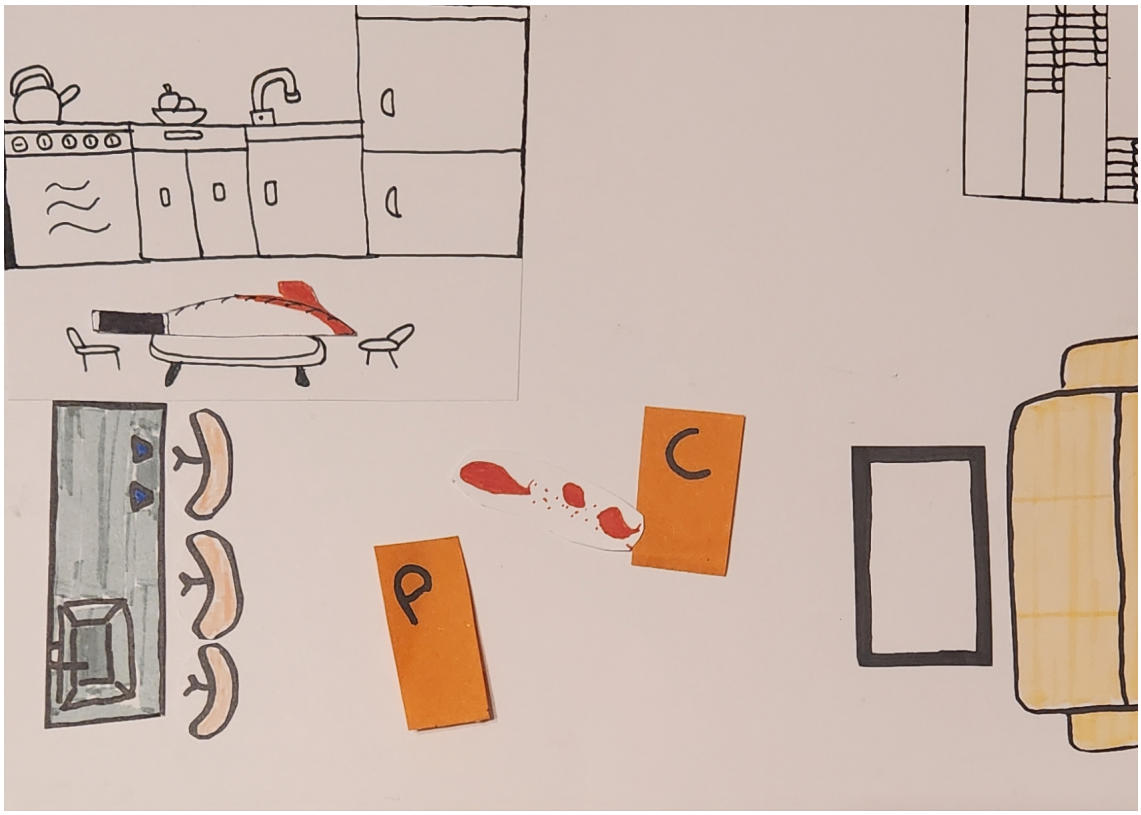


Figure 7.3: Start of game scenario

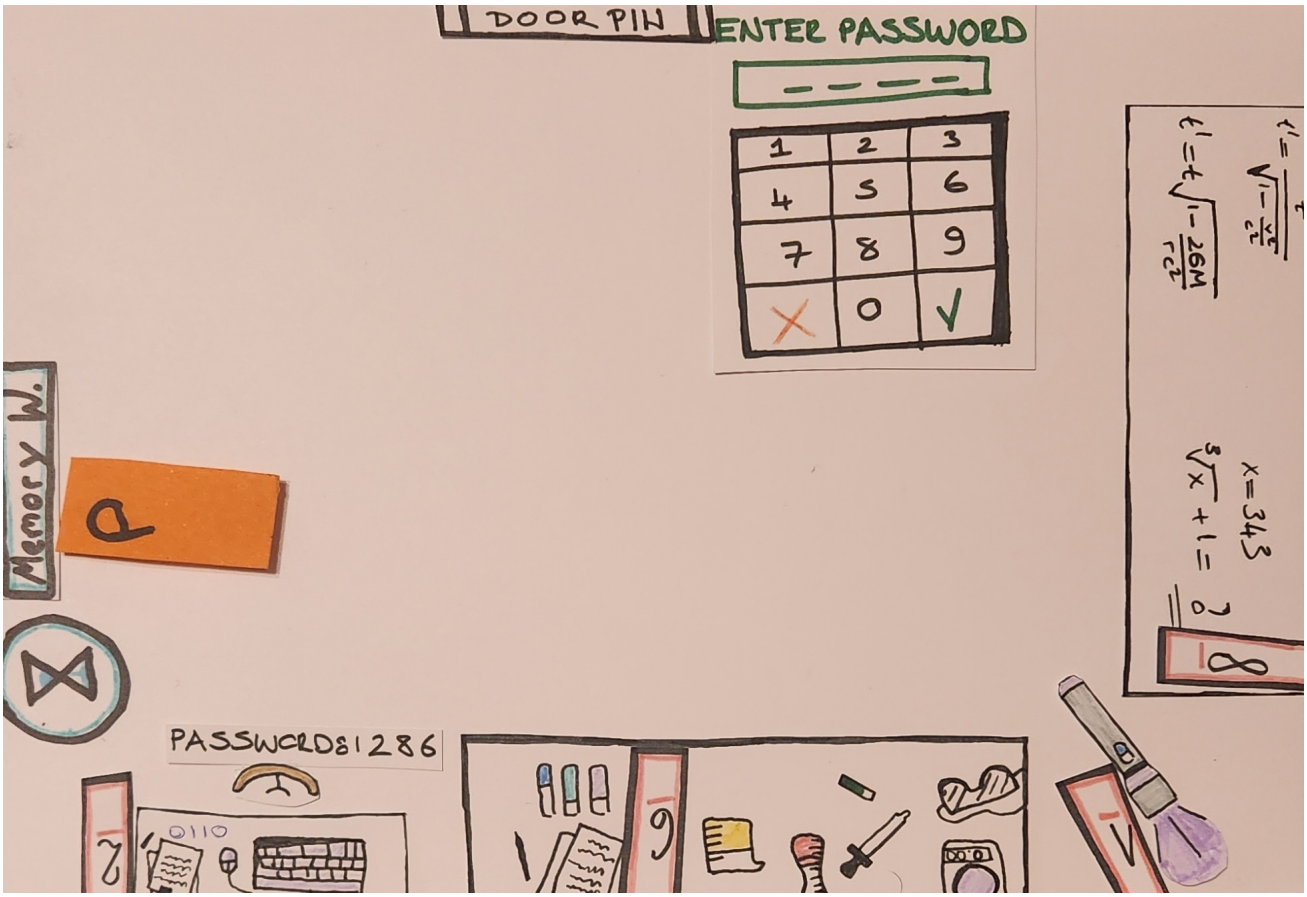


Figure 7.4: Full room with puzzles and memory wall

Chapter 8

Interim Demo

Overview and Task Progression

Entering the Interim Demo phase, our goal was to transition from the paper prototype to a functional digital version of the game that meets the Functional Minimum and Low Target requirements. We successfully implemented the most important core systems: **player movement and interaction, the Time Travel mechanism, the Memory Wall with persistent notes, and the initial framework of the Sanity System.** These systems now provide a stable and reliable foundation for the recursive gameplay we aim to deliver.

While our technical progress is ahead of schedule, the environmental content and level-specific gameplay are still in a preliminary state. The laboratory and break room environments are implemented but currently serve as early versions without full puzzle integration or companion behavior. This was an intentional decision to ensure that all advanced content is built on stable mechanics rather than unfinished systems. With the core features now complete, we expect level design and gameplay integration to progress much faster in the next milestone.

The Layout Design

When we started working on the design for our environment, we kept coming back to the question of what would make the most sense for our story. Because the entire plot revolves around two scientists secretly developing a highly advanced piece of technology, we felt that placing the lab somewhere remote was the most believable choice. We leaned into that "middle of nowhere" feeling—something inspired by places like Area 51, where everything is isolated and difficult to reach. It immediately sets the tone for the player: whatever is happening here is not meant to be found.

Since the protagonists are building a time machine, we also wanted the interior to match the level of technological ambition. A normal-looking lab with chalkboards and worn-down equipment just wouldn't fit. It would feel strange if they were supposedly working on cutting-edge, reality-bending tech but doing so in a space that looked outdated or underfunded. So we chose a high-tech, slightly sci-fi aesthetic—something that feels advanced but still grounded enough to be believable within our world. The goal was to make the player feel from the very first moment that they've stepped into a place where serious, futuristic research is happening.

Figure 8.1: Entrance in the laboratory

The lab itself has two floors. The first floor is designed as the main workspace. This is where most of the machinery will eventually be placed and where the majority of puzzles will take place. We decided to put the time machine and the memory wall on this floor as well, mostly because these two objects are central to the story and to the gameplay loop. Having them easily accessible makes the flow feel more natural, especially since the player will have to engage with them multiple times as they progress.

The second floor has a slightly different purpose. We created a control room filled with computers, screens, and high-tech monitoring equipment. When the player looks out through one of the windows, they can even see a helicopter stationed outside. That small detail helps reinforce the idea that this is a secure, heavily protected facility—again tying back into that Area 51 feeling we want to evoke.

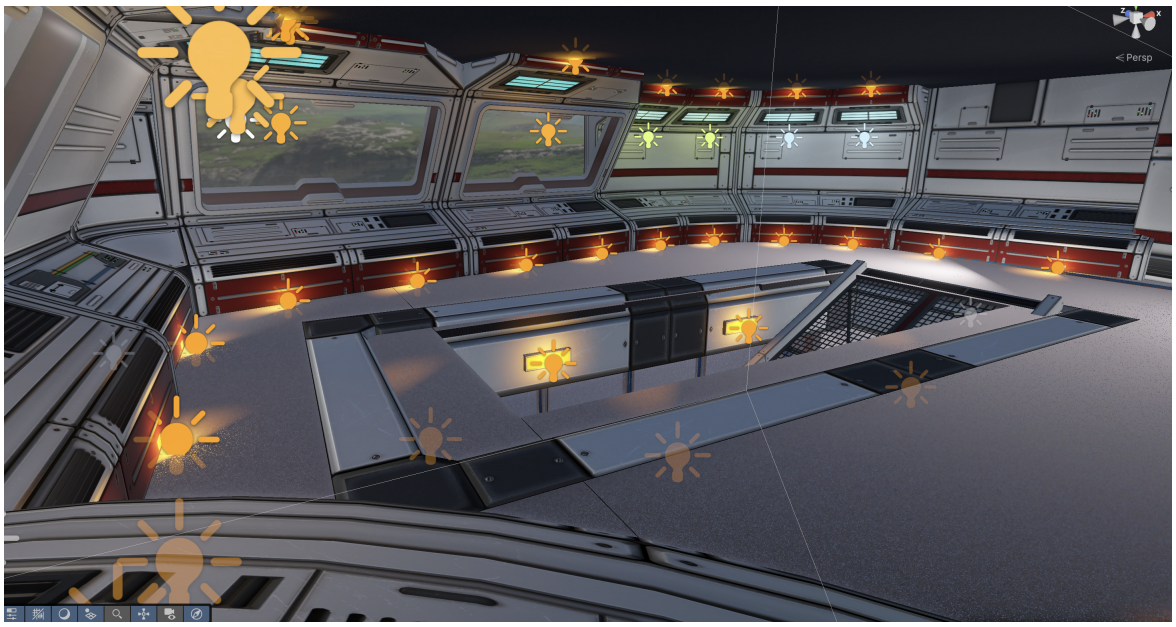


Figure 8.2: Laboratory - Second Floor

Figure 8.3: Laboratory - Second Floor: View Outside

The other room is the break room, which plays a surprisingly important role in the story. This is where the murder reconstruction will happen and where the companion character will be located. Even though it is "just" a break room, we didn't want it to feel empty or rushed. If the scientists are spending long hours—or even days—inside the facility, it only makes sense that the room is fully equipped. So we added things like a kitchen area, a small bar, and some personal touches to make it feel lived-in and believable. It's meant to show that these characters don't just work here; they practically live here.

Figure 8.4: Break Room

Figure 8.5: Break Room: Bar zoomed in

Right now, the laboratory is still pretty empty. Apart from the time machine and the memory wall, most of the props and equipment aren't placed yet. That's intentional—we haven't finalized the puzzles, and we don't want to waste time decorating only to redo everything later. The break room, on the other hand, is about 90% finished. We'll probably add more elements once we work out the details of the murder reconstruction, but it's already close to what we envisioned.

The Time Travel System

To provide the player with an authentic time travel mechanism is one of the main goals for this section of the game. Within the hidden laboratory, the scientists work in, a physical time travel machine exists that can be interacted with by the players. Its metal coating and shiny material will give a more realistic and cold feeling to the touch, receiving its aesthetic direction from other parts and objects inside the laboratory. The size and edges currently shape a cylindrical form with open access to one side, reminiscent of stationary time travel machines from other fictional tales. To further expand on the intuitive design, a player character easily fits inside the machine, and only when doing so can they start jumping into the past. More explanatory and indicative elements, e.g., a path guiding the player towards the machine or color cues to guide the interaction, are to be designed and added at a later point.

Behind the physical attributes of the machine exist two dimensions that build the cohesive structure of our time travel mechanism: authenticity and functionality.

To build a believable experience means to stimulate the players' senses. This is why our time travel machine consists of more than a press of a button. A sound manager and handcrafted sound clips provide the player with a satisfactory feeling when activating the machine from the inside. Secondly, an animator closes the machine's door and returns it to its original state shortly after, locking the player inside during the time travel process. Both the door's movement on activation and its return are made distinguishable and audible by two distinct sounds. While waiting to be released again, the player will listen to another sound that tries to simulate the noises of a futuristic time travel machine, providing more content for the player's ears. Finally, a bright particle effect is displayed inside the machine upon use. This provides greater visual clarity behind the process of time jumping, and again follows futuristic and fictional aesthetics.

The functionality behind the time travel mechanics is modeled through a game manager and a state machine with two separate states, one for the present and one for the past. Initially, the game starts

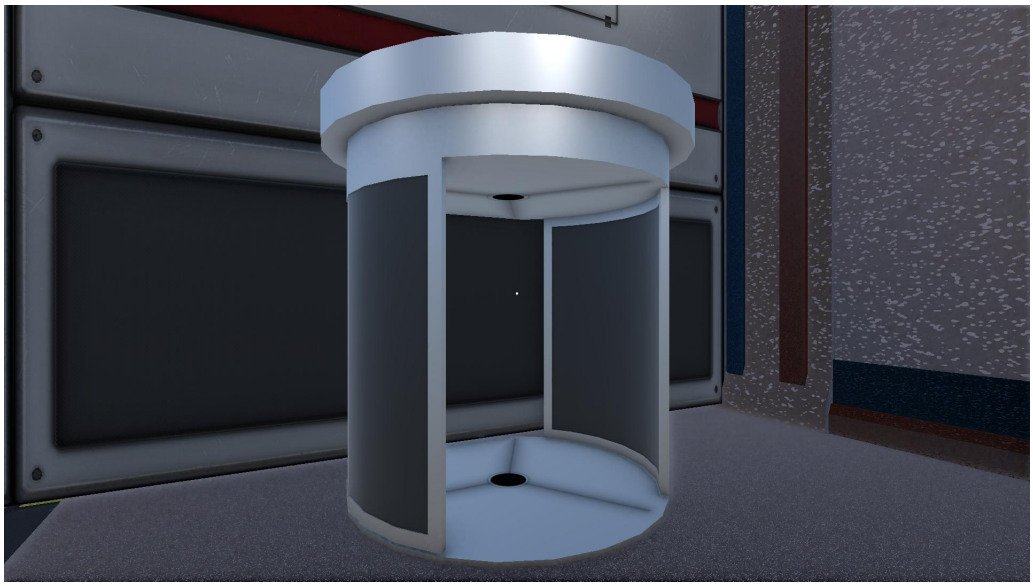


Figure 8.6: The Time Travel Machine

with the game manager instantiating the state machine in the present state, which shows no identifiable features. When activating the button inside the time machine, a method is called to switch the state into the past. As of now, this state can not be left on pure decision, rather a built-in timer will count the time towards the player rejoining into the state of present time. The state-machine structure allows for easy access to enter and exit methods used to implement the necessary changes within our game when traveling back into the past and returning. All objects and information that switch values within a time jump can be stored in the game manager, which also allows for convenient access from other scripts. When the player enters the past, a user interface element appears on the screen that visualizes the amount of time remaining for the player within this loop.

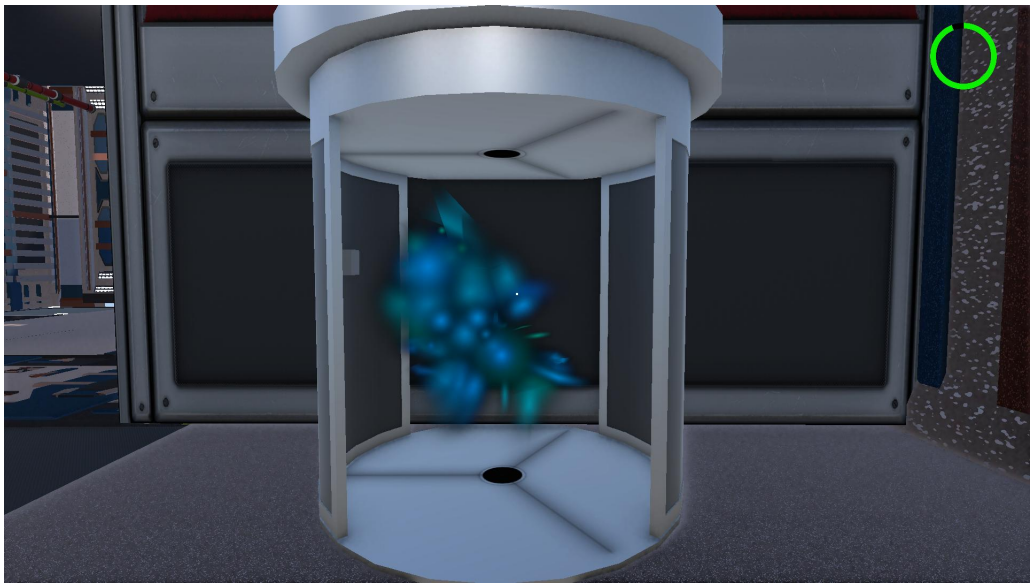


Figure 8.7: The Machine upon use

Memory Wall System

The Memory Wall System is an interactive feature designed to display narrative messages and player-created notes according to the game's loop progression. It serves as an in-game memory archive, helping players keep track of clues, events, and story details across repeated loops. The system combines developer-authored messages with user-generated notes, presenting them in an organized and immersive interface built using Unity's UI Toolkit.

The system begins by loading all available notes from a JSON file located in the project's Resources folder. Each message contains information such as the sender, title, content, and loop requirement. During gameplay, the Memory Wall filters these notes based on the player's current loop level, ensuring that only relevant information is shown at the appropriate time. Notes can be selected with a single click or opened in a detailed view with a double-click, providing intuitive interaction for the player.

Figure 8.8: Notes Panel

In addition to viewing notes, players can create their own entries through the Add Note System. Instead of typing freely, the player constructs messages using categorized word lists—such as actions, places, objects, numbers, and common words. This structured method maintains narrative consistency while still giving players freedom to record meaningful information. When a note is saved, it is appended to the JSON file, and the Memory Wall interface refreshes automatically to display the new entry.
To maintain data integrity and protect narrative content, the system allows deletion only for notes written by the current player. Before a deletion occurs, the system verifies that the selected note's sender matches the player's identity. This prevents accidental removal of developer-authored messages or notes created by other players. Once validated, the system locates the corresponding entry in the JSON file through content matching, removes it, saves the updated file, and refreshes the UI.
From a technical perspective, the Memory Wall System is implemented using Unity's UI Toolkit for interface layout and interactions, JSON serialization for data storage, and C# scripts to manage the UI logic and note operations. The combination of these tools provides a dynamic, persistent, and efficient note-management experience that enhances the game's narrative depth.

As the project evolves, the Memory Wall System is planned to transition from a local JSON-based storage model to a server-based note management system. This upgrade will enable real-time synchronization of notes across multiple devices and players. Instead of relying on local files, all notes both developer-authored and player-generated will be stored, retrieved, and updated through a centralized backend service.

Figure 8.9: Add Note Panel

Figure 8.10: Delete Note Message

The Sanity System

We outline the implementation of the Sanity Manager System, which serves as the core technical backbone for visualizing Amnesiac's mental decline. This architecture ensures the game's psychological elements are modular, scalable, and directly driven by the player's progress across time loops.

The Sanity Manager

The Sanity Manager System is implemented using a decoupled architecture based on the Scriptable Object pattern. This design allows us to define and balance the visual and auditory manifestations of the Amnesiac's madness without modifying the core runtime logic, thus representing a robust and scalable technical achievement. The system is composed of three main components:

- SanitySystemData (Scriptable Object): This is the master data container for all psychological states. It holds an array of SanityLevel objects and manages an internal index. Its primary responsibility is to act as a sequential dispenser and it ensures that new, higher-intensity SanityLevel objects are handed out only when the SanityManager (which tracks the player's time-travel attempts) requests the next level in the predefined sequence.

- SanityLevel (Scriptable Object): This object defines the precise constraints and content for a given psychological state (e.g., "Level 1: Paranoia," "Level 3: Breakdown"). Crucially, it contains a curated, limited array of Visual Effect Enumerations (visualEffects). This array ensures that at lower levels of madness, the player is only subjected to mild, subtle distortions (e.g., a screen jitter), while higher levels unlock more invasive, terrifying effects (e.g., color inversion, geometry warping). This structure decouples the design balancing from the core application logic.
- SanityManager (Runtime Script): This is the central control script, attached to the main Sanity Manager Object in the scene. It holds private references to all available effect implementations (each effect is its own standalone script, such as ChromaticAberrationEffect.cs or AudioDistortionEffect.cs).

Operational Logic and Effect Application

The SanityManager operates on a simple, loop-based logic:

- State Check: On a fixed time interval or based on a significant in-game event, the SanityManager polls the SanitySystemData to fetch the current SanityLevel Scriptable Object.
- Effect Selection: The Manager then accesses the visualEffects array within the retrieved SanityLevel. It uses a weighted random selection algorithm to choose one effect from this permitted subset. This ensures that while only specific effects are available at a given level, the player cannot predict which one will manifest next, maintaining suspense.
- Effect Execution: Once an effect is selected (e.g., the enum for "Texture Bleed"), the SanityManager uses its private member references to call the corresponding method on the specific effect script. The effect script then controls the duration, intensity, and clean shutdown of the visual distortion.

This structure achieves maximum flexibility. By relying on Scriptable Objects, we can iteratively tune the array of effects in each SanityLevel and adjust the frequency of effect application without ever recompiling the core game logic, thus making the system highly adaptable to playtesting and design iteration.

Chapter 9

Alpha Release

9.1 Overview and Task Progression

Entering the Alpha Release phase, our focus shifted from stabilizing individual systems to integrating the main investigative mechanics into a coherent playable structure. Building on the foundation established during the Interim Demo, the core investigation flow starting in the initial room is now supported by functional progression logic and guided player interaction.

The memory system now plays a central role in the investigation, allowing collected clues to be organized through the Memory Wall. In addition, a scene reconstruction mechanic has been implemented to support reasoning about past events based on gathered evidence. Several puzzle systems planned for this milestone have been implemented, including computer-based, physics and biology puzzles, all coordinated through a central game manager that controls game state and progression.

9.2 Scene Reconstruction Framework

9.2.1 System Overview

When the game starts, the player enters a pre-built room from a previous course iteration, which has been extended with the new scene reconstruction mechanics. This space introduces the main gameplay idea: exploring the environment, finding clues, and piecing together what happened in the room.

The player experience focuses on investigation and discovery, but behind the scenes the system relies on a flexible and screen-independent UI setup. All interface elements are built using Unity's `UIDocument` system, allowing the UI to adapt smoothly to different screen sizes and resolutions.

9.2.2 Clue Data Structure

Each clue in the scene is defined using a `ClueData` ScriptableObject. This object stores all information related to a clue in one place, making it easy to add, remove, or modify clues without changing code.

A `ClueData` asset contains:

- the internal monologue shown when the player inspects the clue,
- a short summary that appears in the clue log, and
- a `chronologicalOrder` value that represents the correct order of events.

This chronological value later serves as the reference for checking whether the player has correctly reconstructed the scene.

9.2.3 Interaction Flow and Managers

Several manager components work together to handle clue interaction and progression:

- **Player Interaction:** The `PlayerInteraction` script continuously checks what the player is looking at using a raycast. When a clue is detected, it is highlighted with a visual outline. Interacting with it sends the associated `ClueData` to the `ClueManager`.

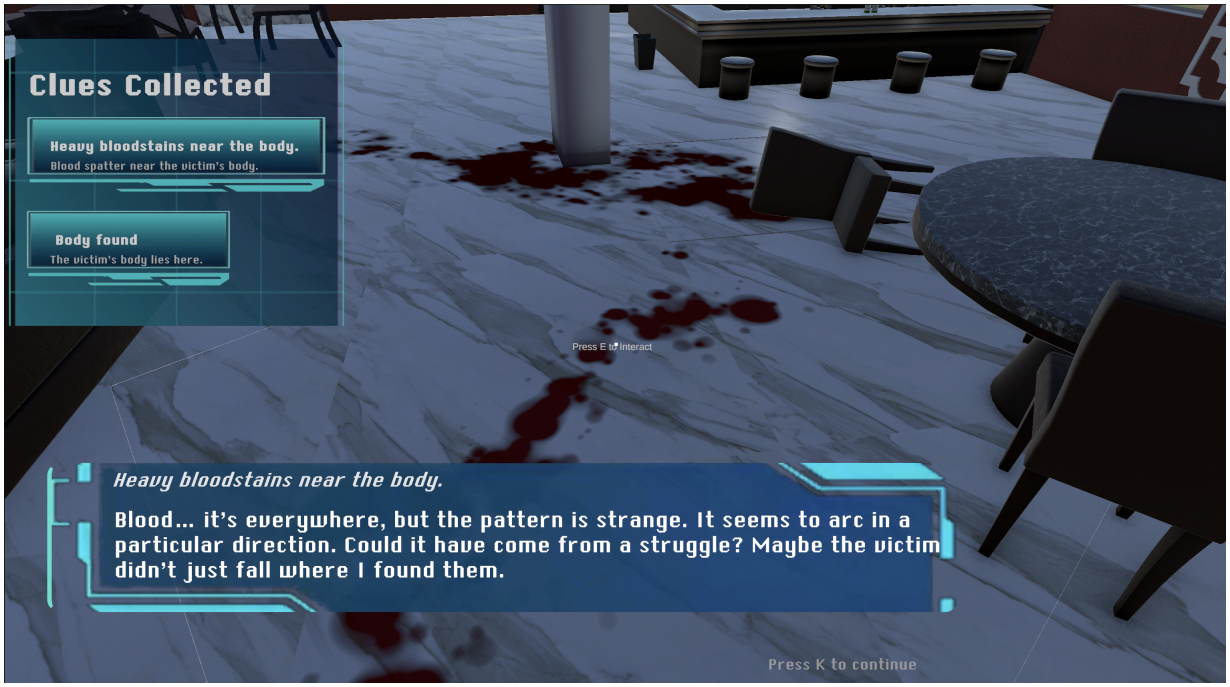


Figure 9.1: Inspecting a clue; Clue Log seen on the left

- **Clue Manager:** The `ClueManager` keeps track of all clues the player has found. It controls the overall flow of the system and makes sure the different UI elements stay in sync.
- **Narrative UI and Log:** When a clue is collected, the `ClueManager` does two things at the same time:
  1. It opens the `InspectionUIController`, showing the player's internal monologue and switching to a cursor-based interaction mode.
  2. It updates the `ClueLogController`, which adds a new entry to the clue log shown on the side of the screen.

This allows the player to immediately understand the meaning of a clue while also keeping track of it for later reference.

Figure 9.2: All Clues Collected

9.2.4 Reconstruction Puzzle

After all required clues have been found, the game transitions to the final puzzle. This is handled by an `IEnumerator` in the `ClueManager`, which opens the `ReconstructionController`.

In this phase, the player sees a board where all collected clues can be placed into slots. The goal is to arrange them in the correct chronological order based on the events they describe.

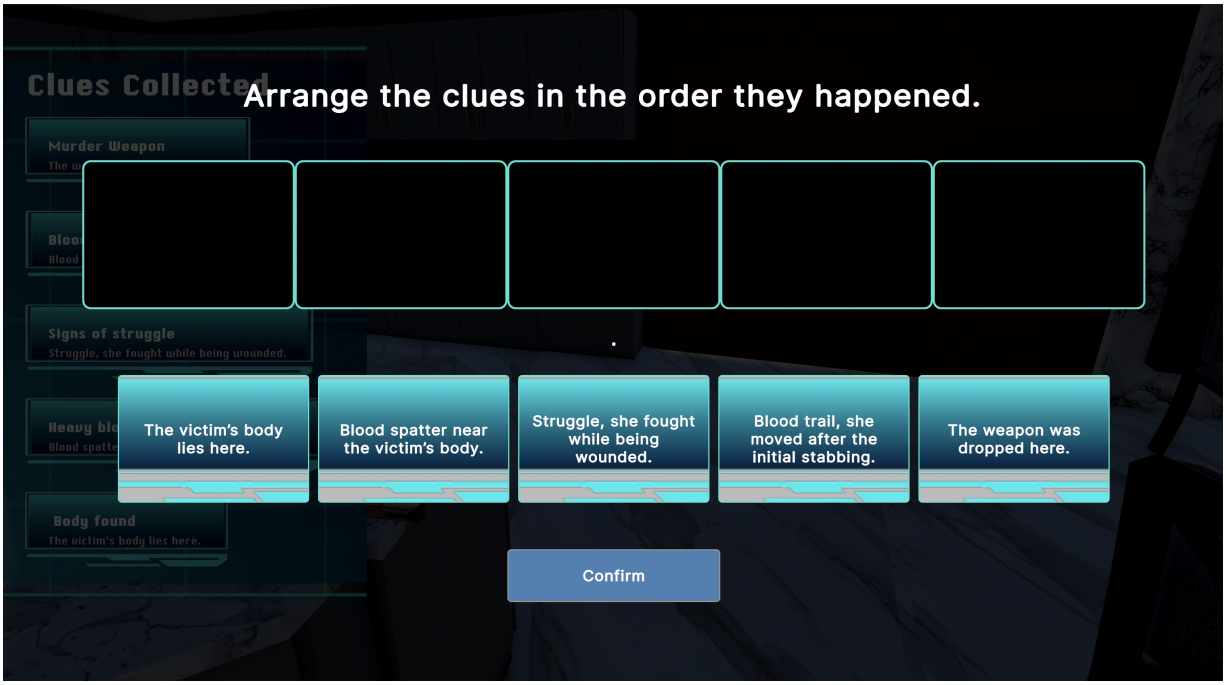


Figure 9.3: Inspecting a clue; Clue Log seen on the left

To check the solution, the system compares the data stored in each slot with the correct order defined

in the `ClueData` objects:

$$\text{Validation} = \sum_{i=0}^{n} (\text{Slot}[i].\text{ID} = \text{RequiredOrder}[i])$$

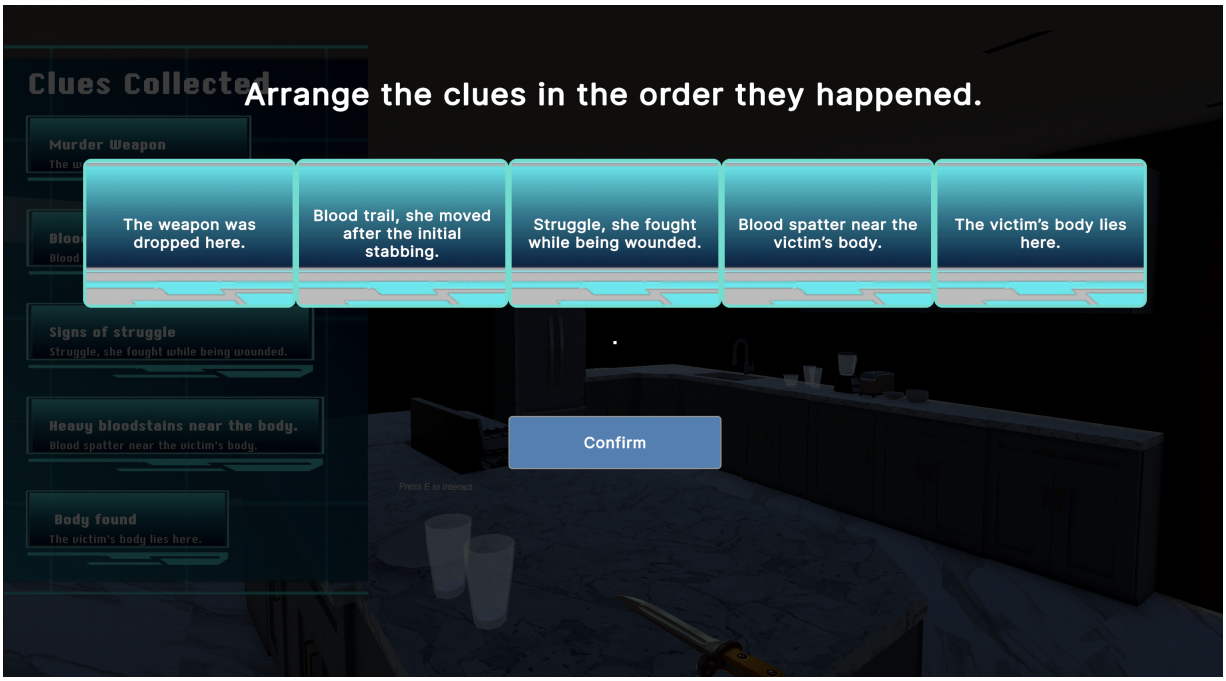


Figure 9.4: The correct order

If the order is correct, the clue log is hidden and the final "Revelation" sequence is triggered. The room's door opens, allowing the player to enter the laboratory area, where the next set of challenges continues the story.

Figure 9.5: The doors are now opened, the player may proceed

9.3 Online Memory Wall System

The Memory Wall System has transitioned from a local JSON-based data model to a server-backed architecture using Supabase. This updates preserves the original functionality of the system while replacing its underlying data storage and synchronization methods. In the previous implementation, all notes were loaded from and written to a local JSON file. Although functional, this approach limited persistence, scalability, and flexibility. With the migration to Supabase, the Memory Wall now retrieves, stores, and manages all notes through a cloud-hosted PostgreSQL database accessed via RESTful API calls. When the user opens the Memory Wall, the system requests the complete note dataset from the server and filters it according to the player's current loop level, ensuring that the narrative pacing and information structure remain identical to the original design. The note viewing experience, including single-click selection and double-click detailed inspection, works same as before, but all data now originates from a centralized backend rather than a local resource file.

The Add Note System follows the same logic. Players still build messages from categorized word lists to maintain narrative consistency, but once a note is assembled it is sent directly to the server instead of being added to a JSON file. The interface updates as soon as the server confirms the entry, making the process feel essentially the same to the player. The deletion rules also continue to work as it did previously: only the creator of a note can remove it, and this check is now performed on the server. This prevents accidental or unauthorized changes and keeps the overall note archive consistent across sessions and devices.

Although the backend logic has been restructured, the appearance and feel of the Memory Wall have been intentionally preserved. New scripts handle communication with Supabase and manage the flow of data between the UI and the server, but the player continues to interact with the feature exactly as before. This transition makes the system more robust and extendable in the long term, enabling persistent storage, cross-device continuity, and even potential multiplayer features, all while maintaining the clarity and usability of its original design.

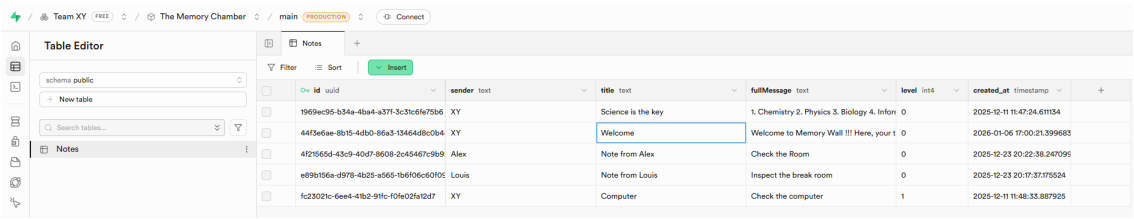


Figure 9.6: Supabase Server Side

9.4 Sanity System Rework

The sanity system was refactored to transition from a monolithic, hard-coded architecture to a modular, decoupled framework. This rework ensures that the system is easily extendible as the project scale increases.

Architectural Shift: Interface-Based Design

The core of the rework involved moving away from a central `switch` statement for effect execution. By implementing the `ISanityEffect` interface, the `SanityManager` no longer requires direct knowledge of specific effect implementations.

- **Decoupling:** Effects are now self-contained classes (e.g., `CameraDistortion`, `CameraSmearPlusHueEffect`) that implement a standardized `TriggerEffect()` method.
- **Dictionary Lookup:** The `SanityManager` utilizes a `Dictionary<VisualEffectType, ISanityEffect>` to map effect enums to their respective components at runtime.

State and Progression Logic

The progression logic was adjusted to decouple numerical sanity decay from visual state transitions.

$$S_{n+1} = \max(0, S_n - R) \tag{9.1}$$

Where S is the current sanity and R is the reduction value. While S decreases with every interaction, the visual `SanityLevel` (stored as a `ScriptableObject`) is only updated every n attempts, where n is defined by the `decreaseLevel` variable.

Enhanced Visual Effects

The primary visual feedback mechanism, `CameraDistortion`, was overhauled to provide a more "psychological" feel. The previous high-frequency vibration was replaced with a smooth drift utilizing Perlin noise for positional offsets:

- **Perlin Drift:** Positional displacement is calculated via $Mathf.PerlinNoise$ to create non-repeating, fluid movement.
- **Interpolation:** All camera transformations and post-processing intensity shifts (Lens Distortion, Saturation, and Vignette) now utilize `Vector3.Lerp` or `Mathf.Lerp` for organic transitions.

9.5 The Companion

The companion system was developed to provide an interactive AI entity capable of following the player and autonomously patrolling a set of waypoints. The system architecture relies on a decoupled parent-child hierarchy to ensure clean movement and animation blending.

System Architecture

The entity is composed of a **Parent Capsule** and a **Visual Model**:

- **Parent Object:** This serves as the physical "brain" and hitbox, containing the `NavMeshAgent`, the custom AI controller script, and a `CapsuleCollider`. The `MeshRenderer` is disabled to make this proxy invisible.
- **Child Model:** A rigged 3D scientist model containing the `Animator`. This separation is crucial for handling animation offsets independently of the physics-driven parent.

Behavioral Logic

The AI logic operates as a finite state machine within the `Update` loop, prioritized as follows:

1. **Wait State:** Upon reaching a waypoint, the agent enters an `isWaiting` state. The `agent.velocity` is manually forced to zero to ensure a clean transition to Idle, while a `Slerp` function rotates the model to maintain eye contact with the player.
2. **Follow State:** If the player is within a defined `followDistance`, the agent prioritizes the player's position as its target destination.
3. **Patrol State:** If the player is out of range, the agent cycles through a pre-defined array of `waypoint` transforms.

Technical Challenges

The most significant challenge during development involved a persistent **Animation Sinking** issue.

Initially, when the companion entered the Idle state, the model would sink into the floor up to its knees. This was diagnosed as a mismatch between the internal "zero point" (root transform) of the specific Mixamo idle clip and the walking animation.

Before identifying the root cause, significant time was spent attempting various technical workarounds, including:

- Adjusting the `Base Offset` within the `NavMeshAgent` component.
- Manipulating the `Root Transform Position (Y)` settings in the animation importer to "Bake Into Pose" based on the feet.
- Implementing a "hard-coded" height correction in the script (`localPosition = new Vector3(0, -0.93f, 0)`) to force the model upwards during specific frames.

Ultimately, the solution was deceptively simple: the original animation clip itself was flawed. By replacing the idle clip with a different version from the library, the height data naturally aligned with the walking rig. This experience highlighted how data-level issues in 3D assets can often masquerade as complex programming bugs, leading to significant troubleshooting overhead for a simple asset-swap fix.

Future Work

While the fundamental movement and animation systems are now operational, the AI is currently in a "groundwork" phase. Future iterations will focus on:

- **Interactive Dialogue:** Implementing a system for the player to engage with the scientist via UI prompts.
- **Dynamic Behavior:** Evolving the companion's logic so that its movement patterns, dialogue, and helpfulness scale according to the player's game progress and discovered milestones.

9.6 Puzzle System

The puzzle system is designed as a central narrative and gameplay mechanic that encourages players to engage with both the Memory Wall and the game's time-loop structure. Progression depends on the player's ability to return to the past, uncover information, and use that knowledge to influence future loops. When the player travels back in time, the door that leads to the next stage becomes locked. To unlock it, the player must solve a series of puzzles, each contributing one digit of a four-digit door code. However, these puzzles cannot be interacted with immediately. The player must first read the relevant note on the Memory Wall; only after reviewing this information is interaction with the puzzle elements enabled. This requirement ensures that the Memory Wall remains an integral part of the experience and that players actively engage with the narrative clues embedded within it.

The puzzle set consists of four distinct challenges, each inspired by a different STEM discipline. In our game, the puzzles are related to physics, biology, and computer science. Rather than being straightforward or procedure-based, these puzzles are designed to promote reasoning, inference, and conceptual understanding. Each puzzle yields one digit of the final code, but the player must determine the correct order through careful analysis of the Memory Wall entries. The notes guide the player not only toward the solution of the individual puzzles but also toward the logical sequence in which the digits must be arranged.

Once the player deciphered the full code, they can enter it into the Memory Wall, where it becomes permanently accessible. This design choice reinforces the cyclical nature of the game's time-loop narrative: although the player may lose progress when traveling back in time, important knowledge such as

the door code persists through Memory Wall entries. As a result, future loops become more efficient, allowing players to bypass previously solved challenges and focus on new objectives.

9.6.1 The Computer Puzzle Game

The computer puzzle is introduced as an interactive terminal that the player can access within the environment. When the player approaches the computer, it becomes highlighted, indicating that it can be interacted with. Pressing the interaction key opens a full-screen terminal interface and switches the player into a cursor-based interaction mode.

The puzzle itself is presented as a corrupted piece of source code implementing the Bubble Sort algorithm. The player is shown several individual code blocks that represent different lines of the algorithm, but these blocks are initially shuffled and separated from their correct positions. Above them, a set of empty slots visually represents the structure of the final program.

The player's task is to drag and drop each code block into the correct slot, effectively reconstructing the Bubble Sort algorithm in the correct order. Each slot only accepts the correct block, preventing random placement and guiding the player toward logical reasoning rather than trial and error. If a block is released in the wrong position, it is returned to the pool.

Once all blocks have been placed correctly, the puzzle enters a completion sequence. A short compilation message is displayed, followed by a visual demonstration of the Bubble Sort algorithm. An example array is shown as vertical bars, which animate step by step as the algorithm sorts the values. This animation reinforces the idea that the player has successfully repaired the broken code.

After the animation finishes, the terminal reveals a digit as a reward, which is one digit of the code to unlock the door leading to the break room.

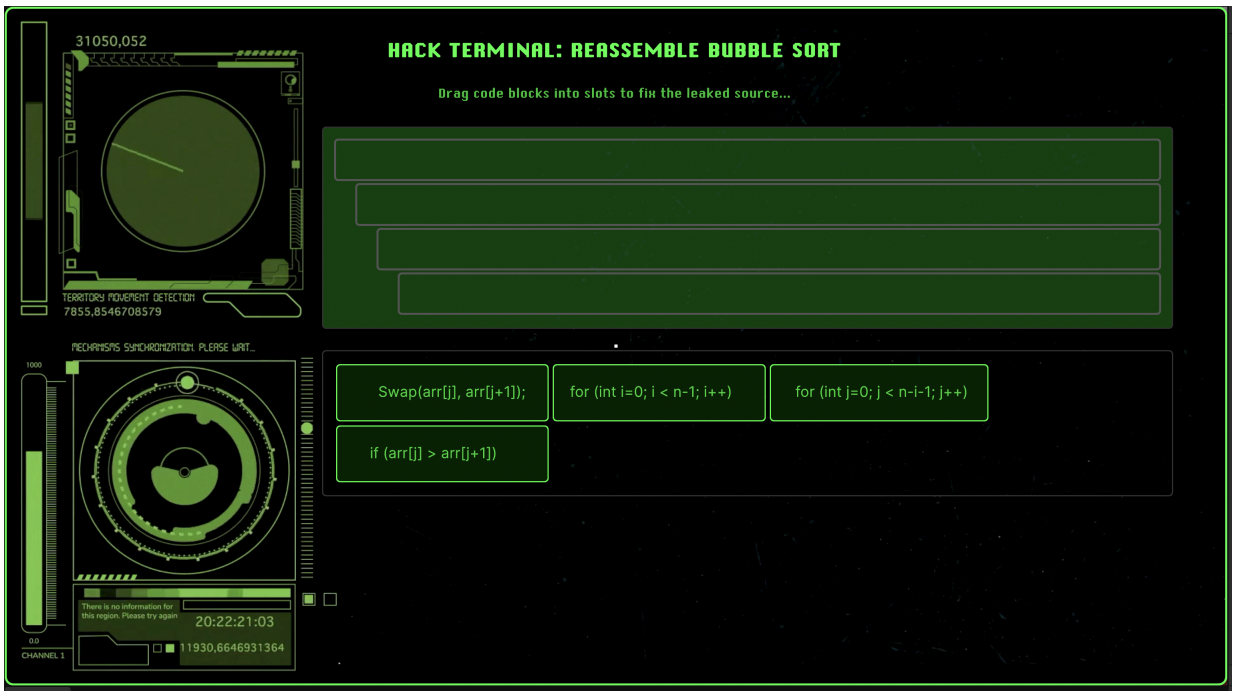


Figure 9.7: Computer puzzle: Start screen

Figure 9.8: Player solves the puzzle

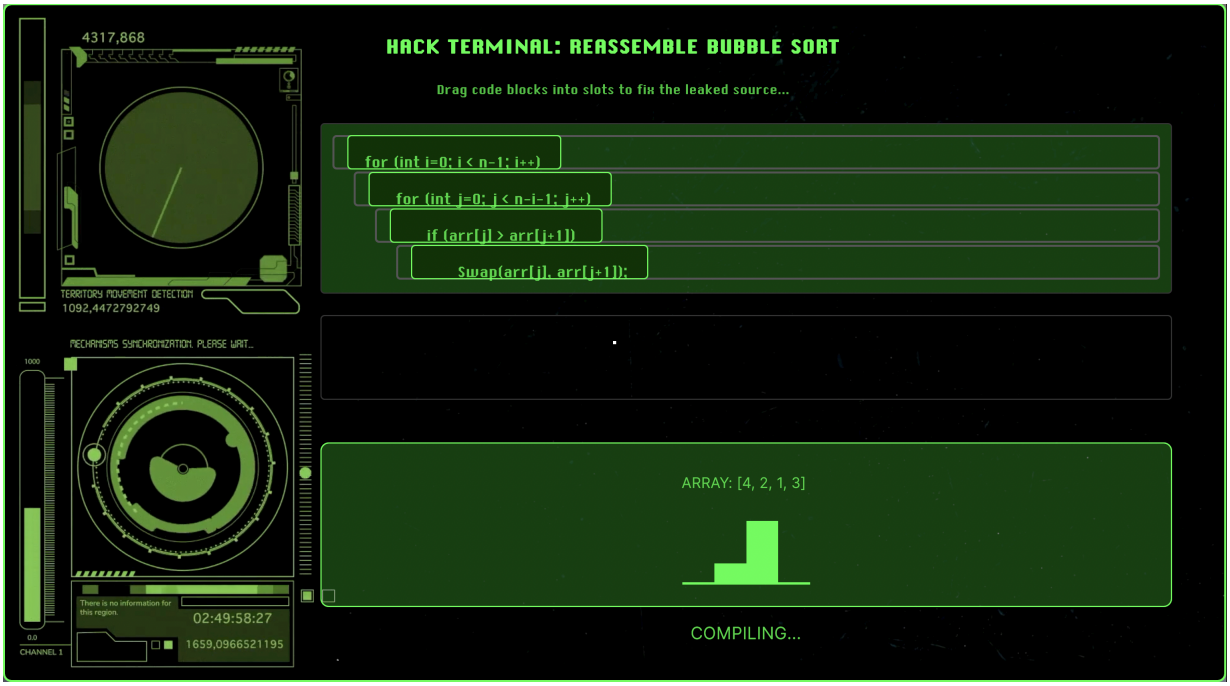


Figure 9.9: Computer puzzle: Player solved the puzzle

Figure 9.10: The digit is revealed

9.6.2 Laser Puzzle

The Laser Puzzle is an interactive environmental challenge where the player must redirect a laser beam from an emitter to one or more receivers to reveal a hidden code. The puzzle logic is built upon a recursive raycasting system that allows for dynamic beam branching and redirection.

System Architecture

The puzzle consists of three primary components:

- **Laser Emitter:** Fires a continuous raycast. If it hit an object with a specific tag (Mirror or Splitter), it invokes a recursive function to generate the next segment of the beam.
- **Directional Mirrors:** Unlike traditional physics-based reflection, these mirrors act as directional hubs. Upon being hit by a laser, they fire a new ray specifically out of their local "forward" face.
- **Beam Splitters:** These components create a "T-junction" effect. When struck, they cast two simultaneous rays perpendicular to their orientation, allowing the player to power multiple receivers from a single source.

Technical Challenges

Several significant hurdles were encountered and resolved during the development of this system:

1. **Non-Uniform Scale Distortion:** One of the primary challenges involved the `LaserTable` parent object, which had a non-uniform scale (e.g., $10 \times 1 \times 5$). When child objects (mirrors) were rotated by 90°, they inherited the parent's stretching, resulting in distorted rectangular meshes. This was resolved by implementing a "Parent-Detach" rotation logic, where the object is briefly unparented, rotated in world space, and normalized to a local scale of 0.4 before being reparented.
2. **Recursive Raycast Self-Collision:** Initially, the laser would "die" immediately upon spawning because it was detecting its own emitter's collider. We implemented a bypass check where the raycast ignores the `gameObject` from which it originated, and added a small offset ($0.1f$) to the starting position of reflected rays to prevent them from getting stuck inside the mirror's own geometry.

3. **Dynamic Inventory and UI:** Managing the puzzle's difficulty required a constraint system. We implemented an inventory tracker that prevents the player from spamming objects. This was integrated with a TextMeshPro UI that updates in real-time as the player places or deletes blocks.

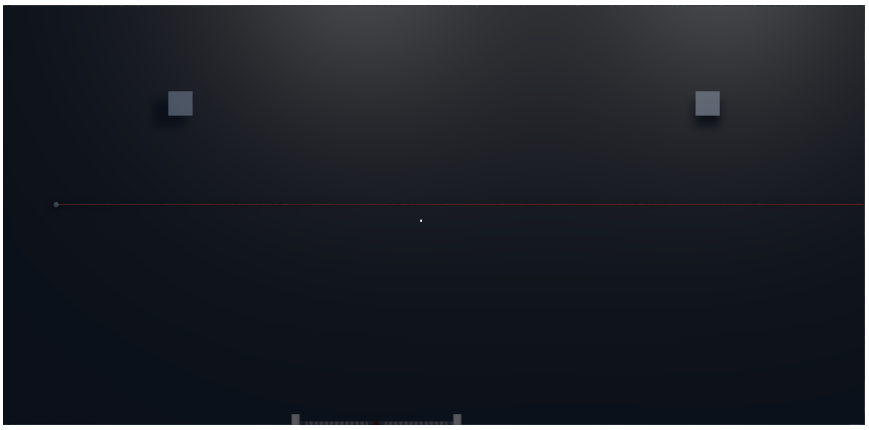


Figure 9.11: Laser Puzzle: Start Screen

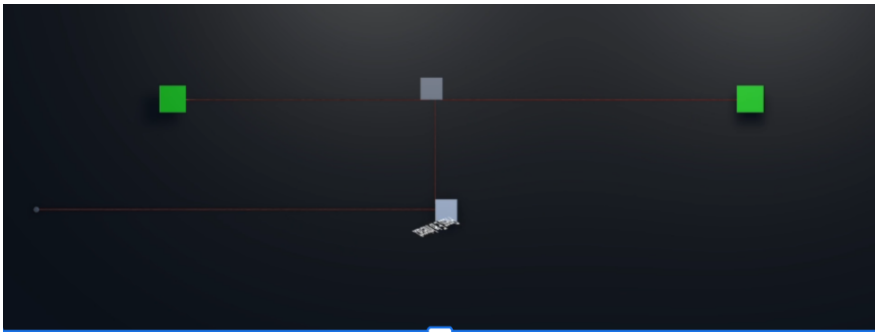


Figure 9.12: Laser Puzzle: Player solved the puzzle

9.6.3 Microscope Puzzle

The microscope puzzle is designed as an observational task built around a virtual microscope interface. When the player approaches the microscope, an interaction prompt appears, and activating the device opens a dedicated inspection UI. Inside this interface, the player can examine a sequence of four microscope images, each displayed initially in a blurred state. Players use the E key to progress through the images and the mouse scroll wheel to adjust the zoom level. A zoom indicator and image index (e.g., Image 1/4) are shown to support orientation during the task.

Each image contains a biological formation composed of microorganism-like visual elements arranged to form recognizable shapes, such as letters. The first slide displays stylized initials ("XY") representing the developer team name figure, while the remaining slides present additional symbols crafted in a similar biological aesthetic (e.g., "C", a reversed "C", and "J"). These formations serve as the core data the player must interpret.

The goal of the puzzle is not to solve a traditional biological analysis, but to identify a numeric value derived from the shapes observed across the four slides. This design encourages careful visual examination and interpretation rather than trial-and-error input. The resulting number contributes directly to the multi-step code required to unlock a sealed door in the game environment.Interaction with the microscope is permitted only after the player interacted Memory Wall.

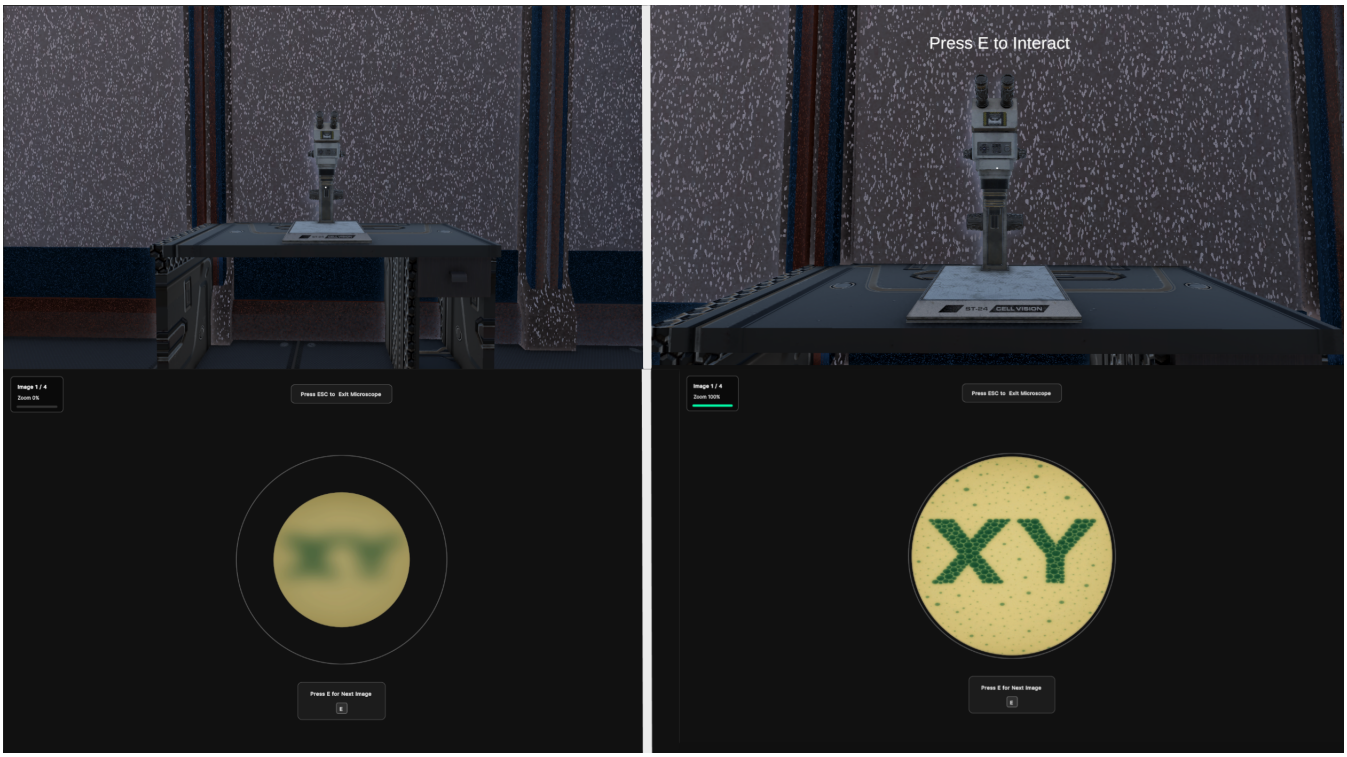


Figure 9.13: Microscope: Interaction and Interface

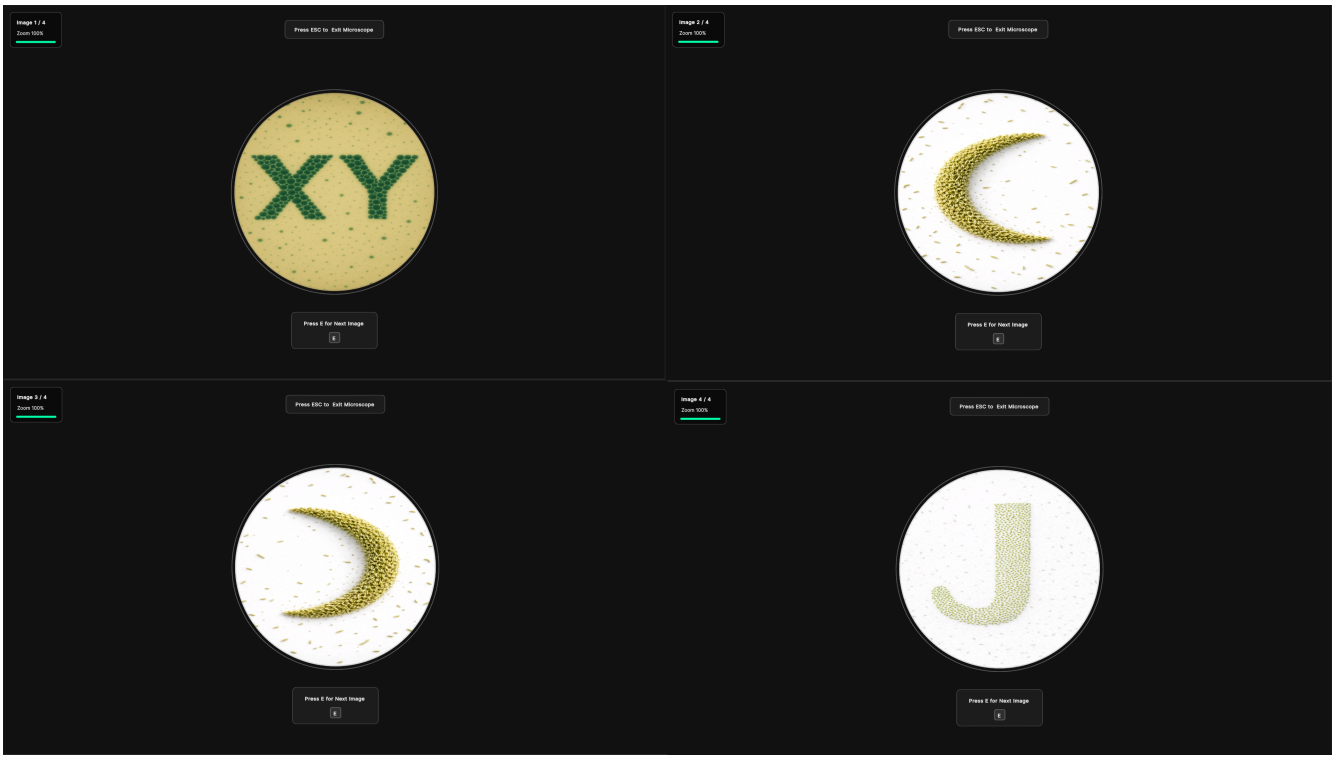


Figure 9.14: Unblurred Images

9.6.4 Flashlight Puzzle

The flashlight puzzle is a search-and-follow task that directs the player around the laboratory to find the hidden digit. Clues lie hidden right in front of the player's eyes. To make the clues visible, the player must pick up a special flashlight; only then can they begin to follow the trail of clues. This puzzle is designed to encourage passive exploration. As the player moves through the lab, they naturally look

around, observe their surroundings, and absorb the atmosphere of the space they are in.

Once collected, the flashlight is held in the player's hands and can be carried freely. It emits a bright, colorful cone of light that interacts with mysterious artefacts concealed on the floors and walls of the laboratory. When the player shines the light onto these hidden objects, they gradually become visible, revealing images that guide the player toward the next location. By following each revealed clue, the player is eventually led to a final artefact. This last discovery points them toward the location of one digit of a door code. After understanding the riddle, the player can retrieve the required digit and return the flashlight to its original place.

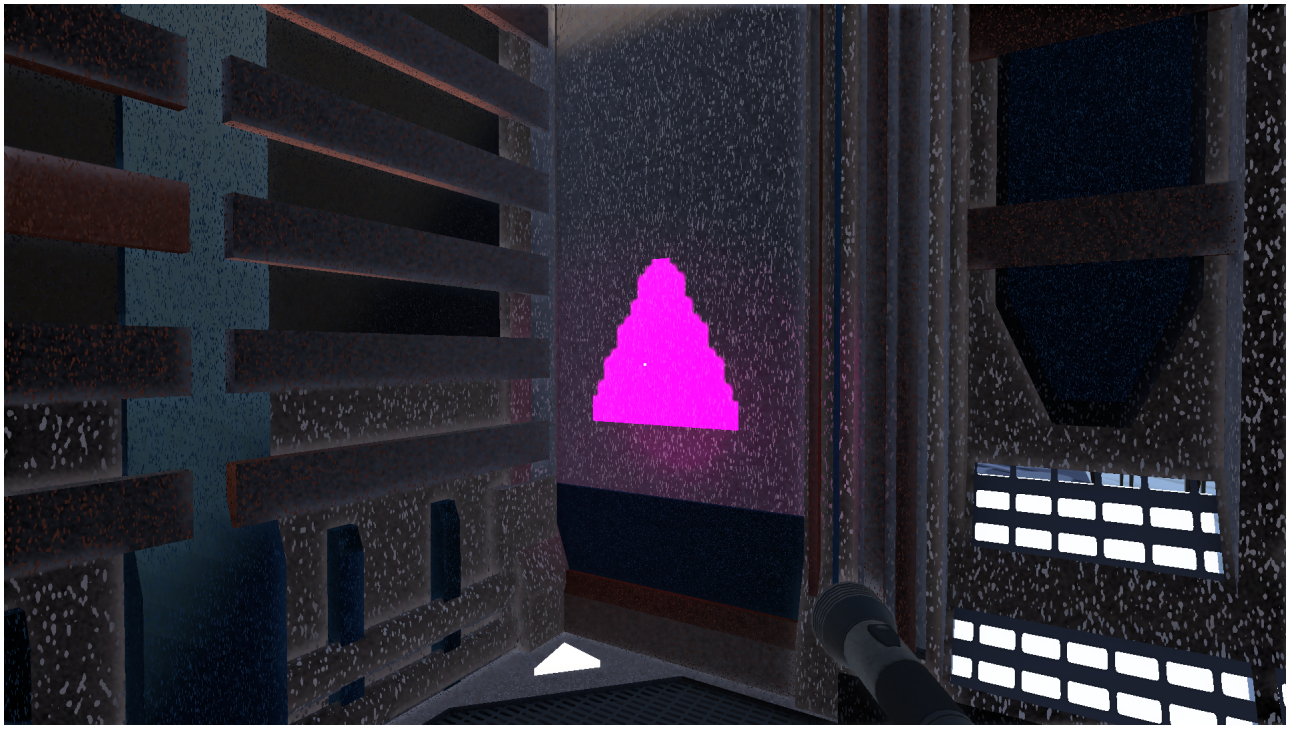


Figure 9.15: A clue becoming visible through the flashlight

To support this mechanic, a custom material was created for the hidden clues. The material is based on a specialized unlit shader, developed using Unity's Shader Graph system. It uses the position, direction, and rotation of the flashlight to generate distance and angle masks. The artefacts only become visible when the flashlight is aimed correctly within these masks. As the player moves, the flashlight continuously updates its position and direction, allowing the shader to dynamically control the visibility of the hidden clues.

Here, P_{world} denotes the world position of the pixel, L_{pos} represents the position of the light source, and R defines the maximum range. Regarding the angle calculation, δ represents the dot product between the normalized direction vector pointing towards the pixel and the light's forward direction L_{dir}. The spot angle is denoted by θ, while ϵ (set to 0.1) defines the softness of the cone's edge. Both masks are then multiplied via cross product to create the visibility factor for the albedo value of the material's color attributes.

Calculation of the Distance Mask (M_{dist})

$$\vec{v} = P_{\text{world}} - L_{\text{pos}} \tag{9.2}$$

$$d_{\text{rel}} = \frac{\|\vec{v}\|}{R} \tag{9.3}$$

$$M_{\text{dist}} = 1 - \text{saturate}(d_{\text{rel}}) \tag{9.4}$$

Calculation of the Angle Mask (M_{angle})

$$\delta = \frac{\vec{v}}{\|\vec{v}\|} \cdot L_{\text{dir}} \tag{9.5}$$

$$M_{\text{angle}} = \text{smoothstep}(\theta, \theta + \epsilon, \delta) \tag{9.6}$$

Figure 9.16: Mathematical derivation of the flashlight shader logic.

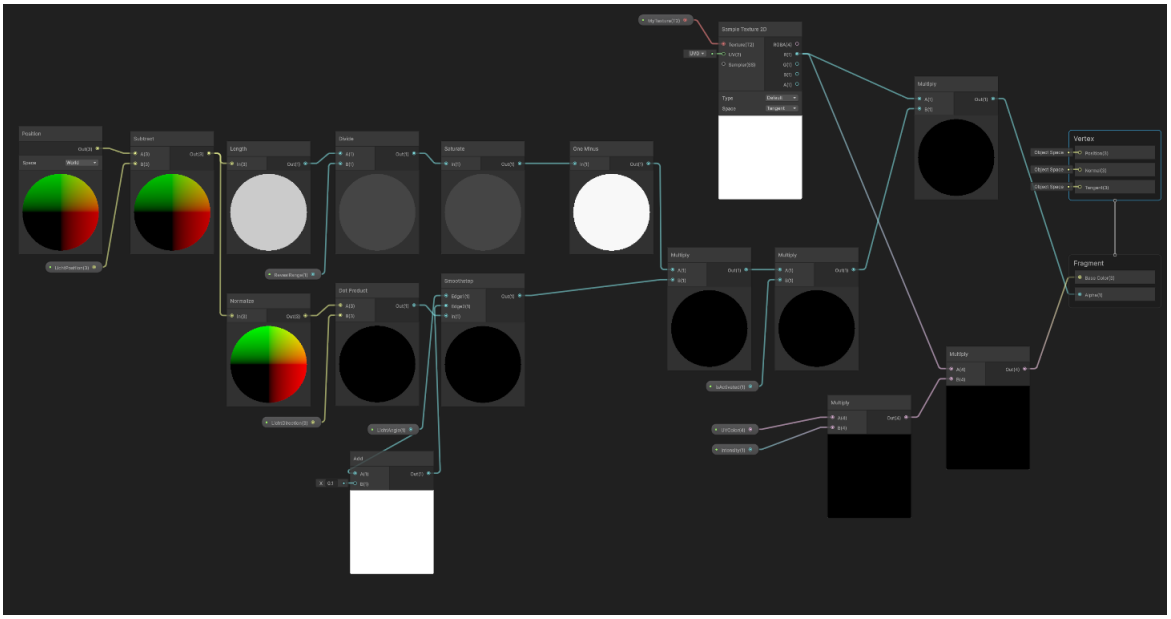


Figure 9.17: The view of the shadergraph

One challenge that was approached lies in Unity's uniform handling of material bases. Multiple textures are needed to display every part of the puzzle inside the game. Yet, only one texture can be applied in the texture graph and stay visible while playing. To easily avoid this problem, multiple materials are created using the shader graph. Each material can hold its individual texture. This way, only one shadergraph was implemented and utilized for multiple materials, allowing for various versions of hidden clues again.