Learning Sign Language Through Puzzle-Solving Game

BY

NG JIA MIN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my tutor, Mr. Luke Lee Chee Chien, for being my mentor, monitoring my progress and giving me valuable advice.
In addition, I would like to thank my parents and family for their love, support and continuous encouragement throughout my study.

ABSTRACT

This project focuses on developing an innovative survival-adventure escape game, Hushh, designed to integrate and teach sign language through an interactive gaming experience. By immersing players in a virtual world where sign language is the primary mode of communication, the game cleverly addresses the communication barriers faced by the deaf community. In Hushh, players find themselves in a forest environment, interacting with non-playable characters (NPCs) using sign language, solving puzzles, unlocking doors, and advancing through the game. This immersive experience aims not only to raise awareness about the communication challenges faced by the deaf but also to provide a fun and engaging way for players to learn basic sign language.

The game's development follows gamification principles, making the learning of sign language more interactive and enjoyable compared to traditional methods. By combining survival elements with sign language learning, the game encourages empathy, helping players understand the daily communication challenges faced by the deaf community, while offering a firsthand experience of those barriers. Hushh is not just an entertainment platform; it also serves as a tool for promoting social inclusion, helping players gain a deeper understanding of deaf culture and communication methods.

In terms of sign language creation, the game will utilize animation software to custom-design sign language gestures, ensuring precise and expressive sign language representation. This approach enhances the game's interactivity and educational value, contributing to its dual purpose of entertainment and learning.

Area of Study: Sign Language Learning, Game Animation Production
Keywords: Sign Language, Survival-Adventure, Game-Based Learning, Immersive Experience, Deaf Community Awareness

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.0 Background information

In daily life, communication mainly relies on spoken language, which is convenient for the general population, but for deaf people, it can be a source of frustration. Furthermore, from kindergarten to university education, except for special schools, other regular schools do not teach sign language, resulting in a lack of public awareness of this important communication method. This situation indirectly causes a lack of education and awareness about deaf individuals and sign language in society, relegating sign language to communication only between deaf-mute people, who are gradually being overlooked in the broader social environment.

Although Malaysia's social welfare system is relatively well-developed, with low-cost education, healthcare, and official services, deaf people theoretically should be able to enjoy these public services. However, in practice, they often do not receive adequate support in these services. For example, hospitals, police stations, and government agencies lack staff specifically trained to assist the deaf, and communication largely depends on spoken language. This makes it extremely difficult for deaf people to access services. In addition to the inconveniences in daily life, deaf people also face significant challenges in employment. Due to a lack of good learning environments, many deaf people are illiterate, which to some extent hinders their career development. Globally, the difficulty of employment for deaf people is not uncommon. Companies are often unwilling to hire applicants who have not received a good education, let alone the deaf community.

Just like learning any other language, it is not necessary to master all grammar and vocabulary to communicate effectively; one only needs to learn some basic expressions. The same goes for sign language—learning basic sign language expressions allows for effective communication. For example, in situations like ordering food at a restaurant, visiting a hospital, or shopping, a deaf-mute person can clearly express their needs using sign language. If society could pay more attention to the deaf community and improve understanding of their situation, lifestyle, and methods of communication, it would help improve their living conditions and promote better social integration.

In addition, today's market is filled with gaming apps, ride-hailing apps, and shopping apps, but there are very few apps related to sign language, and most of them are translation apps. For example, apps like "Sign Language Dictionary" or "ASL Dictionary" provide instructional videos on sign language and vocabulary, along with cultural background information about sign language. According to App Annie, user reviews for sign language learning apps have been generally positive, with an average user rating of 4.5 stars in 2023, particularly for their comprehensive teaching content and ease of use. Some VR platforms have also started creating applications for gesture recognition and sign language interaction. These applications detect users' movements with haptic gloves to recognize sign language, offering a novel experience for users. Analysis has shown that using VR for educational applications enhances learning outcomes and helps students maintain better focus and memory **[1].** The advantage of these applications lies in providing rich and complete sign language learning resources, making them very user-friendly for beginners. VR games allow users to interact in virtual environments using sign language, and users can learn the details of sign language through haptic gloves.\

1.1 Problem Statement and Motivation

Figure 1.1 Statistics of Hearing Loss Cases in Malaysia

Hearing loss is the third most common physical illness after arthritis and heart disease**[2].** According to a 2019 report by the World Federation of the Deaf, there are approximately 70 million deaf people worldwide. In Malaysia, data from the Social Welfare Department (JKM) indicate 41,819 registered deaf individuals, though this likely underrepresents the actual population **[3].** These statistics, which are two years outdated, underscore the persistent neglect of the deaf community. Despite Malaysia's seemingly comprehensive welfare system, deaf individuals encounter significant barriers in daily life. Hospitals, police stations, and government agencies predominantly lack professional sign language interpretation, with communication heavily reliant on verbal language, thereby impeding deaf people's access to

essential services **[4].** Educational limitations have consequently rendered many deaf individuals functionally illiterate, severely restricting their professional prospects **[5].** Globally, deaf employment rates remain low, with minimal corporate inclusivity further marginalizing this community **[6].**

Currently, sign language education remains confined to traditional classroom settings, characterized by low engagement and limited public participation. This project proposes leveraging gamification to create a more compelling and interactive approach to sign language learning. By lowering educational barriers and enhancing societal understanding, the design aims to foster communication and integration between deaf and hearing populations. Beyond mere entertainment, the ultimate objective is to challenge social exclusion, cultivate a more inclusive environment, and facilitate more equitable social participation for the deaf community.

1.2 Project Objectives

This project aims to create an innovative escape adventure game named **"Hushh"**, blending survival game mechanics with sign language learning, creating a unique interactive experience. The core goal is to integrate sign language seamlessly into the gameplay, allowing players to solve puzzles, communicate with NPCs, and overcome challenges using sign language.

The project focuses on developing a game that combines entertainment with educational value. By immersing players in the survival elements of the game and enabling them to interact with NPCs through sign language, it helps players gain a deeper understanding of how sign language is applied in daily life. The game will serve as a tool to break down the communication barriers between hearing individuals and the deaf community, raising awareness of the challenges faced by the deaf in society.

According to Michael Erard (2017), a letter from the University of Washington stated that deaf individuals are expected to constantly adjust to the same level as hearing individuals **[7]**. This raises the question: why can't hearing individuals adjust for the deaf. Gamescom Asia's 2020 statistics show that there are approximately 25 million gamers in Asia **[8].** In a survey of Malaysian players, it was revealed that Malaysia’s 20.1 million gamers spent an impressive $673 million on games in 2019, making it one of the biggest gaming markets in Southeast Asia **[9].** This demographic is the target audience for this game, and since few games of this type

exist, it represents an innovative opportunity. This shows that gaming is an important business worth investing in, with a competitive edge

It is important to note that this project does not focus on developing in-depth linguistic courses or teaching advanced sign language skills. Instead, the goal is to make sign language accessible and enjoyable, using the game's narrative and interactive tasks to introduce basic sign language concepts. The game is not intended to replace formal sign language education but rather to serve as an introductory platform, sparking interest and reducing the stigma associated with sign language, encouraging its integration into everyday life.

1.3 Project Scope

Hushh is a survival-adventure escape game designed to teach and incorporate sign language. The game blends traditional survival mechanics with sign language challenges, offering players a fun yet educational experience. Below are the game's design concepts:

1. **Sign Language-Based Gameplay:**
    - Most interactions in the game must be completed using sign language, such as communicating with NPCs, solving puzzles, and unlocking password locks. Players need to rely on sign language without the ability to speak.
    - The game will guide players through learning basic sign language, such as common greetings, words, and phrases.
2. **Intense Survival Environment:**
    - The game features mutated creatures and other threats, where players must stay calm and use sign language to communicate with other characters in order to avoid detection. To stay quiet and safe, players must use sign language to avoid mutant creatures that are sensitive to sound.
    - The core gameplay involves escape tasks, where players unlock sign language-based password locks, decipher puzzles, and progress through the storyline.
3. **Enhanced Interactivity and Learning:**

- Various levels in the game involve not just solving puzzles, but also interacting with NPCs using sign language. As players progress, they will apply and refine their sign language skills.
- The game's storyline is conveyed through sign language and visual clues, providing players with a deeper understanding of how sign language is used in real-life situations.

4. **Education Through Entertainment:**
    - The game's design integrates storylines and interactive tasks that enable players to enjoy the gameplay while learning sign language. This approach seamlessly combines education and entertainment, appealing to a broad audience, including younger players, while making sign language learning engaging and fun.

Through these designs, Hushh not only offers a thrilling survival adventure but also serves as a tool for sign language education. It helps players understand and appreciate the challenges of the deaf community, indirectly teaching them sign language as they navigate through puzzles, exploration, and interactions in the game.

1.4 Contribution

Hushh combines entertainment with social impact, offering a unique contribution to both the gaming and education industries. The core aim of the game is to provide a fun and interactive platform that introduces sign language to a wider audience. By engaging players in immersive survival scenarios, the game will enhance their understanding and empathy for the challenges faced by the deaf community. Players will not only learn basic sign language but will also experience the difficulties faced by people who rely on sign language as their primary means of communication.

The impact of this project is profound. It has the potential to bridge the communication gap between hearing and deaf individuals, making sign language more accessible and normalized in everyday life. By incorporating sign language into the game environment, **Hushh** will inspire players to continue learning and practicing sign language beyond the game, fostering inclusivity and improved communication. The significance of this project lies in its ability to

leverage the global appeal of gaming to promote education, inclusivity, and awareness, leading to lasting positive social change.

Additionally, **Hushh** will not only benefit individuals interested in learning sign language but also has the potential to serve as an innovative example for future educational games. By seamlessly combining language learning with gameplay, the project demonstrates how gamification can be used to teach essential social skills in an engaging and enjoyable way. While the project may currently have some limitations as it is being developed by an individual, the novel approach it introduces will undoubtedly attract further attention from professionals, pushing forward the development of sign language education. This makes the project not only relevant to gamers and educators but also highly desirable, opening new doors for sign language education and creating a lasting impact.

1.5 Report Organization

This report is structured into seven chapters to present the development of the HUSHHH sign-language escape game in a clear and logical sequence:

Chapter 1 – Introduction

Provides the project background, problem statement, objectives, scope, and contributions. It explains the motivation for combining sign-language education with an immersive escape-puzzle game.

Chapter 2 – Literature Review

Reviews related technologies, platforms, and existing sign-language learning applications. It highlights their limitations and identifies the research gap that this project aims to address.

Chapter 3 – System Methodology / Approach

Describes the overall development methodology and system model. It includes the system design diagram, system architecture diagram, use case diagram, and activity diagrams, outlining the conceptual flow of the game.

Chapter 4 – System Design

Details the complete design of the game system, including the system block diagram, key components, circuits (if applicable), and interaction operations. It explains how input, processing, and output layers integrate to deliver real-time gameplay.

Chapter 5 – System Implementation

Presents the technical implementation, covering hardware setup, software setup, configuration, and operation of the game. It also discusses development challenges and solutions.

Chapter 6 – System Evaluation and Discussion

Reports the testing methods, performance metrics, and user evaluation results. It discusses how the final system meets the objectives stated in Chapter 1 and reflects on issues encountered during development.

Chapter 7 – Conclusion and Recommendation

Summarizes the overall achievements, key findings, and educational contributions of the project, and proposes possible future enhancements such as expanded levels, mobile deployment, and community integration.

Chapter 2

Literature Review

2.1 Review of The Technologies

2.1.1 Hardware Platform

The development and testing of the project were carried out on a laptop computer with the following specifications:

Component	**Specification**
Model	ASUS Vivobook 14 OLED (Model: M3401QC)
Processor	AMD Ryzen 7 5800H with Radeon Graphics @ 3.20 GHz (8 cores, 16 threads)
Graphics	NVIDIA RTX Studio Laptop GPU, 4 GB dedicated VRAM
Memory	16 GB DDR4 (15.4 GB usable, 3200 MT/s)
Storage	477 GB SSD (approx. 450 GB available)
Display	14-inch OLED panel
Operating System	Windows 11 Home Single Language, Version 24H2, 64-bit

Table 2.1.1 Hardware Platform Specifications

2.1.2 Firmware/OS

The project runs entirely on a Windows-based environment and does not require any dedicated firmware.

The development and testing were performed using the following operating system:

Item	Specification
Operating System	Windows 11 Home Single Language (Version 24H2, 64-bit)
Build & Updates	Latest cumulative updates applied as of project completion
Firmware	No custom firmware required; standard ASUS BIOS supplied with the laptop

Table 2.2 Operating System and Firmware Specification

2.1.3 Database

This project is a stand-alone Unity game, so it does not exchange data online and does not require any external database server. All data needed for gameplay is packaged inside the Unity build and loaded locally at runtime.

Unity Scenes & Prefabs – Level geometry, UI layouts, triggers, NavMesh information, and interactive objects (doors, NPCs, countdown zones, etc.) are embedded directly in the scene or prefab files.

Scriptable Objects / Inspector Serialization – Parameters such as enemy damage values, countdown lengths, door-open settings, and audio/video references are stored as serialized fields ([SerializeField]) on MonoBehaviour components and saved with the scene..

No Network Layer – Players can run the executable completely offline; all resources (video files, audio clips, textures, fonts) are bundled with the build.

Because every parameter—whether it is a trigger range, a countdown duration, or a video file path—is stored locally, no SQL/NoSQL database or cloud service is needed to play or maintain the game.

2.1.4 Programming Language

The entire gameplay is scripted in C#, Unity's primary and fully integrated language. C# offers strong typing, object-oriented structure, and direct access to Unity's APIs for physics, animation, UI, audio, and video, which makes it ideal for this project.

Key usage examples drawn from the codebase:

Player Controller (player)

Handles W-A-S-D movement, smooth rotation, running, slope detection, and step-climbing using Rigidbody physics plus raycasts. Tracks collectibles (diamonds) and raises events (onDiamondCollected) to update the UI.

Game State & Health (PlayerHealth)

Manages health values, plays damage sounds, and shows a Game Over video when health reaches zero. Provides a ForceGameOver() method so other systems—such as a countdown timer—can trigger the same death/video sequence.

Countdown & Timed Failure (CountdownGameOverTrigger)

Starts a ten-minute countdown when the player enters a zone. Continuously updates an on-screen Text element and calls PlayerHealth.ForceGameOver() when time expires.

Environmental Interaction

Doors & Puzzles (PressKeyOpenDoor, PressKeyOpenDoorWithMist)—open doors on specific key presses, optionally spawn poison mist, play sounds, and damage the player for wrong input.

Zone UI Controller

Displays multi-step computer-screen interfaces, deducts countdown time when a special button is pressed, and plays a one-shot audio cue for that action.

VictoryTrigger

Stops the main camera, plays a victory video, and optionally loads the next scene after the clip finishes.

Audio/Video Integration

AudioSource objects play UI clicks, damage sounds, countdown-deduction effects, and environmental audio. VideoPlayer components handle intro, death, and victory cut-scenes with Prepare, loopPointReached, and errorReceived callbacks.

UI System

Built with UnityEngine.UI (Buttons, Scrollbars, Text). Uses event listeners (button.onClick.AddListener) and coroutines (IEnumerator) for smooth fading, countdown updates, and asynchronous video preparation. Apart from standard Unity shaders, only minimal built-in ShaderLab/HLSL materials are used for lighting and surface effects; no custom shader code was required.

2.1.5 Algorithm

The key algorithms in this project focus on player control, countdown management, enemy AI, environmental interaction, and multimedia cut-scene flow. All logic is implemented in C# scripts inside Unity, using event-driven design and coroutines to provide smooth real-time interaction.

1. Player Movement and Physics

- **State-Machine Control**
    - An enum MovementState { Idle, Forward, Backward } manages the player's idle, forward, and backward states.
    - Update() reads W/A/S/D and Shift key inputs to switch states and trigger the correct Animator parameters (walk, run, idle).
- **Smooth Rotation**
    - Target rotation is updated from input, and Mathf.LerpAngle interpolates between current and target angles for natural turning.
- **Slope and Step Detection**
    - Physics.Raycast detects ground normals and calculates the angle to Vector3.up to determine slope steepness.

- When within a valid range, an extra forward force assists uphill movement.
- Additional forward and upward raycasts detect step height; if within the threshold, an upward force is applied to climb the step automatically.

2. Health and Damage Handling

- **Area Damage (DamageTrigger)**
  - OnTriggerStay checks if the player remains inside a danger zone.
  - A time stamp nextDamageTime ensures damage occurs at set intervals, calling PlayerHealth.TakeDamage() and playing a damage sound.
- **Death and Forced Game Over (PlayerHealth)**
  - When health reaches zero, player controls and the main camera are disabled, a Game Over video is played, and the Game Over UI is shown.
  - The ForceGameOver() method lets other scripts (e.g., countdown) trigger the same death sequence.

3. Countdown System

- **Zone-Based Activation**
  - CountdownGameOverTrigger starts a coroutine Tick() on OnTriggerEnter, decreasing remaining time with Time.unscaledDeltaTime.
  - UI text updates every frame in MM:SS format.
- **External Time Reduction**
  - CountdownGameOverTrigger starts a coroutine Tick() on OnTriggerEnter, decreasing remaining time with Time.unscaledDeltaTime.
  - UI text updates every frame in MM:SS format.
- **Timeout Action**
  - When the countdown reaches zero, it automatically calls PlayerHealth.ForceGameOver() to trigger the Game Over video

4. Enemy AI and Environmental Interaction

- **Enemy Chasing (MyEnemy)**
    - Continuously measures distance to the player.
    - If within chase range and under the max limit, the NavMeshAgent destination is set to the player; otherwise, the enemy returns to its start point.
    - Animation mode switches between Idle, Run, and Attack based on NavMeshAgent.velocity.
- **Danger NPC (dangerScript)**
    - Monitors distance to the player; when close, the NPC rotates to face the player and triggers an alert animation.
    - After the animation finishes, it returns to Idle and resumes monitoring.
- **Interactive Triggers**
    - Door with Poison Mist (PressKeyOpenDoorWithMist) – Inside the trigger, pressing the correct key (I) opens the door and hides specific objects. Incorrect input (if "Poison Mist" is enabled) spawns a mist effect, plays an error sound, and reduces player health.
    - Victory Trigger (VictoryTrigger) – Disables the main camera, plays the victory video, and after the video shows a win panel or loads the next scene.

5. Multimedia and Cut-Scene Flow

- **Intro Sequence (IntroFlowController)**
    - Three stages: static image → intro video → final image/start button.
    - Coroutine PlayIntroVideo() prepares and plays the video, with optional Space-key skip.
    - When the video ends or the skip key is pressed, it fades to the final panel and starts background music.
- **Safe Video Playback**
    - All VideoPlayer components call Prepare() before Play(), and use loopPointReached/errorReceived callbacks.

- After playback, RenderTexture is released to prevent frame ghosts.

6. UI Events and Audio

- **Button-Driven Interaction**
  - All UI buttons use Button.onClick.AddListener, including the countdown-deduction button, door controls, and retry button.
  - Clicking the Deduct Time button immediately calls deductSfx.Play(), ensuring the sound plays only on that action.
- **CanvasGroup Fades**
  - Smooth transitions between images and videos use CanvasGroup.alpha and coroutines for fade-in/out effects.

2.1.6 Summary of the Technologies Review

The overall technology stack for this Unity-based game is compact, fully self-contained, and offline-ready, combining standard PC hardware, a Windows operating system, and Unity's integrated toolchain to deliver all gameplay features without external services.

Hardware & OS – Development and testing were completed on a Windows 11 laptop with an AMD Ryzen 7 CPU, NVIDIA RTX GPU, 16 GB RAM, and SSD storage. No special peripherals or firmware are required; the final build runs on any modern Windows PC with comparable specifications.

Database – No external database or network layer is used. All data—scene geometry, triggers, videos, audio clips, and gameplay parameters—are stored locally inside Unity scenes, prefabs, and serialized fields. Scriptable Objects and inspector serialization handle configuration values, allowing the entire game to run completely offline.

Programming Language – All gameplay logic is implemented in C#, Unity's primary scripting language. C# provides strong typing, object-oriented structure, and direct access to Unity APIs for physics, animation, UI, audio, and video playback, enabling concise and maintainable code.

Algorithms – Core algorithms cover player movement physics, enemy AI navigation, countdown management, damage/health tracking, and multimedia cut-scene control. These are

implemented through Unity event triggers, coroutines, and state machines, ensuring smooth real-time interaction and responsive gameplay.

Multimedia & UI – Unity's built-in VideoPlayer handles intro, death, and victory sequences with safe preparation and cleanup. UnityEngine.UI components (Buttons, Scrollbars, Text) drive all interactive menus and countdown displays, while CanvasGroup fades provide polished visual transitions.

2.2 Review of the Existing Systems/Applications

2.2.1 Lingvano ASL

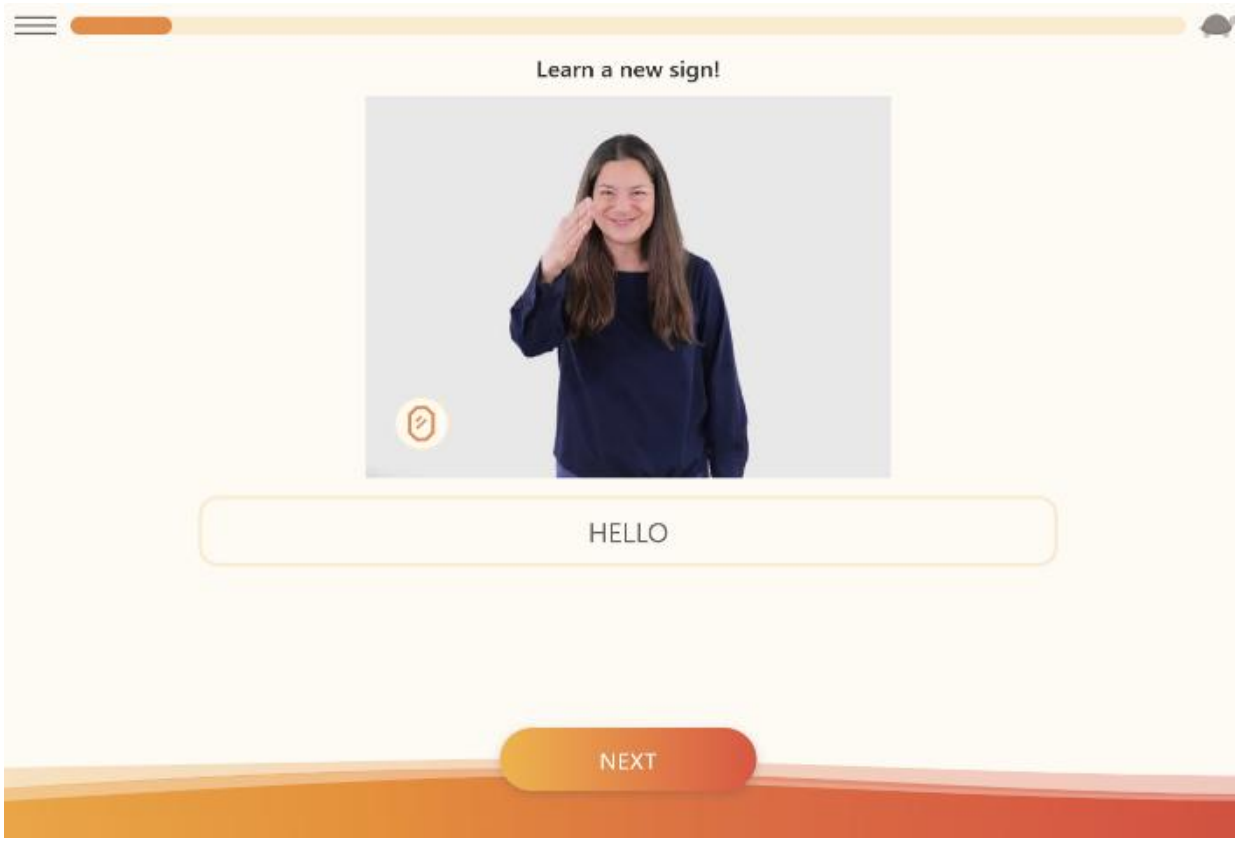


Figure 2.2.1 Lingvano ASL

Lingvano ASL is a popular sign language learning platform that focuses on helping users learn sign language through video-based lessons. The platform offers multiple language options, allowing users to choose from three different types of sign language, including American Sign Language (ASL), German Sign Language (DGS), and International Sign Language (IS). This diverse selection enables learners from different countries and regions to find content tailored to their needs.

The core advantage of the platform lies in its use of engaging sign language videos for instruction. Users can learn sign language gestures and expressions by watching these videos, making the learning process both intuitive and accessible.

2.2.2 Sign School

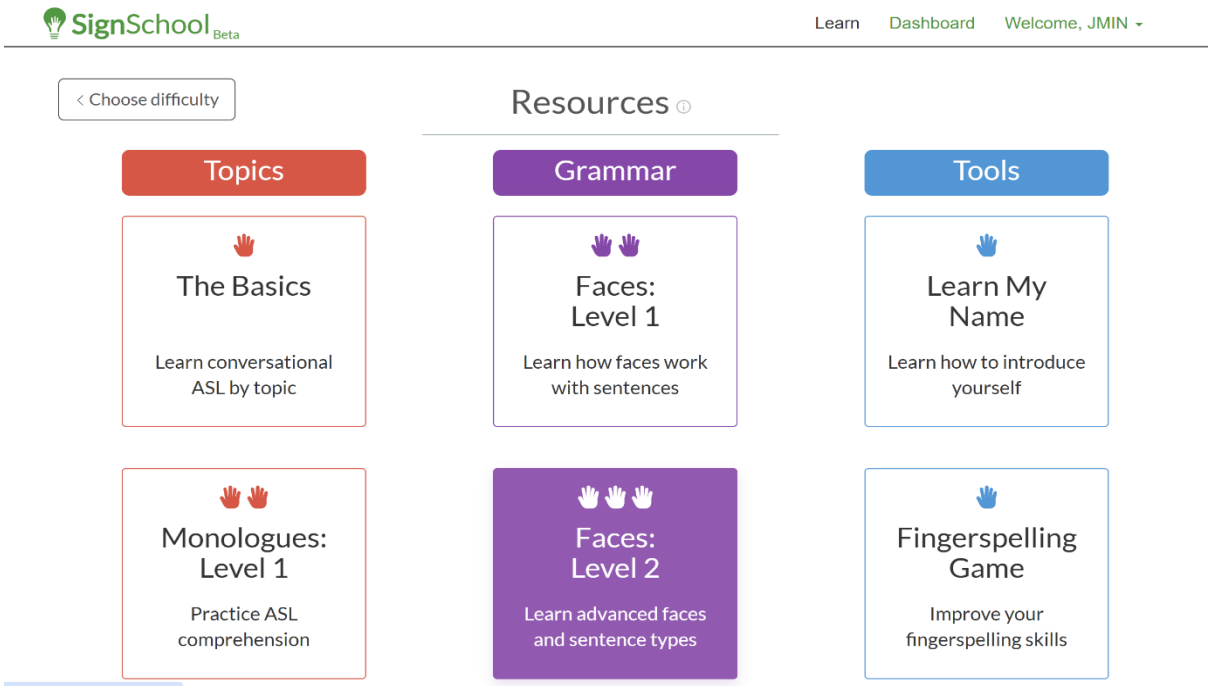


Figure 2.2.2 Sign School

SignSchool is an interactive platform for learning American Sign Language (ASL). It offers a range of lessons and tools, including topic-based learning, grammar exercises, and interactive games. Users can improve their skills through structured courses on ASL basics, facial expressions, and fingerspelling. SignSchool provides a flexible and engaging way for learners of all levels to master ASL at their own pace.

2.2.3 Lifeprint.com

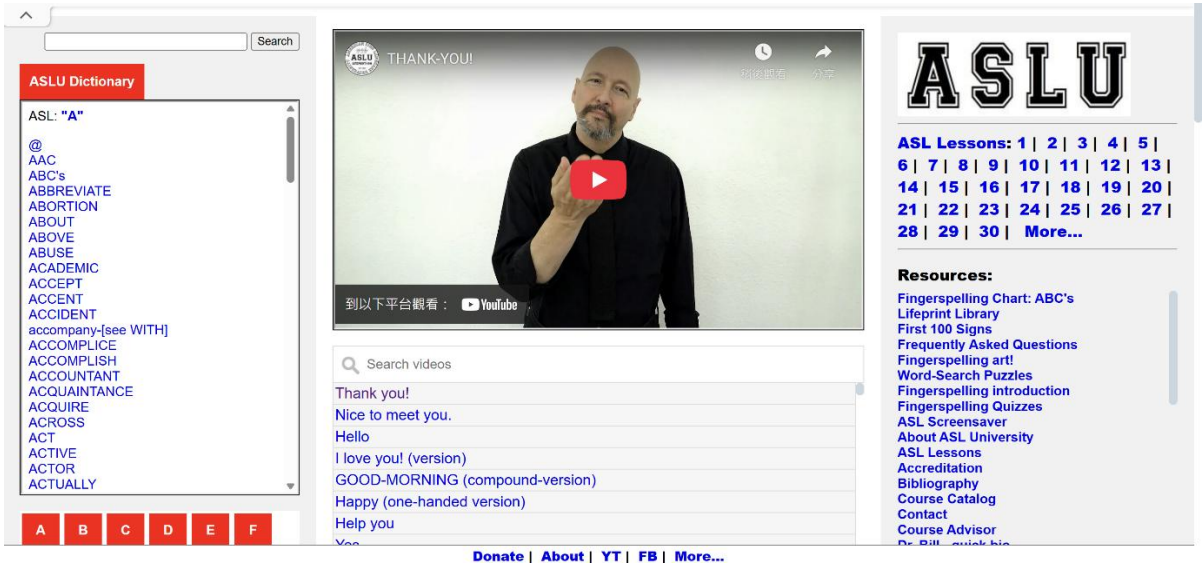


Figure 2.2.3 Lifeprint.com

Lifeprint.com is an online platform dedicated to teaching American Sign Language (ASL) through videos, interactive lessons, and an extensive ASL dictionary. It offers resources for learners at all levels, including fingerspelling practice, video tutorials, and helpful tools. The site provides a structured learning path and supports users with additional resources like puzzles and quizzes. Lifeprint is a comprehensive and accessible tool for anyone interested in learning ASL.

2.2.4 Limitation of Previous Studies

Lingvano's advantage lies in allowing users to repeatedly watch sign language learning videos, which can adapt to learners' varying speeds, providing high flexibility. This approach is particularly suitable for learners at different paces, helping them practice and reinforce their sign language skills according to their own needs. However, the video tutorial-based learning method is similar to reading a book, lacking interactivity. Over time, this can lead to boredom among learners. While effective in the initial stages, the lack of real-time feedback and practical opportunities may cause learners to lose interest. Additionally, compared to other platforms, Lingvano presents relatively simple content, with a lack of interaction and innovation, making it less competitive.

SignSchool's competitive advantage lies in its comprehensive teaching approach. The platform offers learners courses of different difficulty levels, allowing them to choose learning content based on their progress and abilities. This systemized course design helps learners gradually build their knowledge of sign language, from basic to advanced skills. By dividing the learning process into manageable stages, SignSchool ensures that each phase is thoroughly understood. However, despite offering rich content, SignSchool also suffers from a lack of interactive tools. After completing video tutorials, learners lack real-time interaction with other learners or instructors, missing the opportunity for immediate feedback and correction, which can affect their ability to apply what they've learned in real-life situations.

LifePrint, like Lingvano and SignSchool, uses video-based teaching methods to help learners master American Sign Language (ASL). The platform's ASL dictionary and foundational courses are very helpful, but it faces the same issue of having a monotonous learning method. Although LifePrint provides abundant video resources and sign language vocabulary teaching, its reliance on self-paced video learning lacks interactivity. Learners are unable to receive real-time feedback or personalized guidance, which makes it harder for them to apply what they've learned in real-world situations. Compared to other platforms, LifePrint's learning content is relatively fixed and repetitive, offering limited dynamic learning experiences, which restricts its appeal and practicality in the long run.

The above website are typical educational websites that invariably attract only a small group already interested in sign language. Our project aims to create an innovative platform that breaks through these limitations, using game-based approaches to further stimulate the curiosity and empathy of ordinary people

Across **Lingvano**, **SignSchool**, and **LifePrint**, several shared weaknesses emerge:

- **Lack of Interactivity and Real-Time Feedback**

All three rely on video-based instruction. Learners can watch and review at their own pace but cannot receive immediate correction or guidance, making it difficult to apply skills in real situations.

- **Monotonous Learning Experience**

The tutorial style resembles reading a book. Over time this static format leads to boredom and reduced motivation.

- **Relatively Simple and Repetitive Content**

Compared with other options, the material remains fixed and lacks innovation, offering limited variety and long-term appeal.

- **Limited Reach Beyond Existing Enthusiasts**

These sites mainly attract users already interested in sign language, failing to engage a wider audience.

2.2.5 Proposed Solution

Enhancing Interactivity Through Gameplay

This project combines sign-language learning with level-based gameplay. As players progress through increasingly challenging stages, they naturally encounter and use sign language. Victory or defeat provides immediate feedback on accuracy, removing the need for traditional step-by-step error correction. Stage objectives and puzzle clues supply continuous external motivation, sparking both competitiveness and curiosity. Compared with the repetitive question-and-answer drills of a conventional classroom, this approach significantly reduces learning fatigue and keeps players engaged in a lively, enjoyable environment.

Story-Driven Sign-Language Content

Every sign-language action is embedded in the main storyline and in character interactions, allowing learning to unfold alongside narrative progression. The plot not only creates a meaningful linguistic context but also makes sign language integral to solving puzzles and advancing the story, avoiding the monotony of rote imitation. Branching dialogue and interactive scenes help players link knowledge with context, improving retention and real-

world transfer. Ongoing updates to the storyline and characters can further maintain freshness and encourage players to return for repeated exploration.

Sustained Long-Term Engagement and Broader Reach

The game's structure supports regular additions such as new levels, seasonal events, and special challenges, continually extending the learning content and maintaining long-term appeal. While designed for sign-language learners, its entertainment value also attracts a general gaming audience, allowing them to encounter sign language naturally as they play. Through carefully designed tasks and immersive experiences, players gain first-hand insight into the communication challenges faced by the deaf community, fostering empathy and deeper understanding. This blend of education and entertainment provides a sustainable path for spreading sign-language knowledge and raising public awareness.

2.2.6 Comparison between Existing System and Proposed System

Aspect	**Lingvano ASL**	**SignSchool**	**LifePrint.com**	**Proposed System**
Learning Method	Allows users to repeatedly watch sign language learning videos, adapting to different speeds and providing high flexibility.	Offers courses of different difficulty levels, including topic-based lessons, grammar exercises, and interactive games.	Uses video-based teaching methods with abundant video resources and an ASL dictionary.	Combines sign-language learning with level-based gameplay where victory or defeat provides immediate feedback on accuracy.
Interactivity	Video tutorial-based learning similar to reading a book,	Despite rich content, **lacks interactive tools** and **real-**	Relies on self-paced video learning and **lacks**	**Enhancing interactivity through gameplay**;

	lacking interactivity and **real-time feedback**.	**time interaction** with learners or instructors.	**interactivity**, with no real-time feedback or personalized guidance.	stage objectives and puzzle clues motivate players and remove the need for traditional error correction.
Experience	Effective in the initial stages but may lead to boredom; relatively simple content and a lack of innovation reduce competitiveness.	Systemized course design helps gradual learning but lacks immediate feedback and correction, affecting real-life application.	Content is relatively fixed and repetitive, offering limited dynamic learning experiences and long-term appeal.	**Story-driven sign-language content** embedded in the main storyline and character interactions avoids rote imitation and keeps learning fresh.
Target Audience	Attracts a small group already interested in sign language.	Primarily for learners progressing through ASL levels.	Appeals to those specifically seeking ASL resources.	Designed for **sign-language learners and general gamers**, spreading knowledge and fostering empathy toward the deaf community.

Table 2.2.1 Comparison between Existing Systems and Proposed System

Chapter 3

System Methodology/Approach

3.1System Design Diagram/Equation

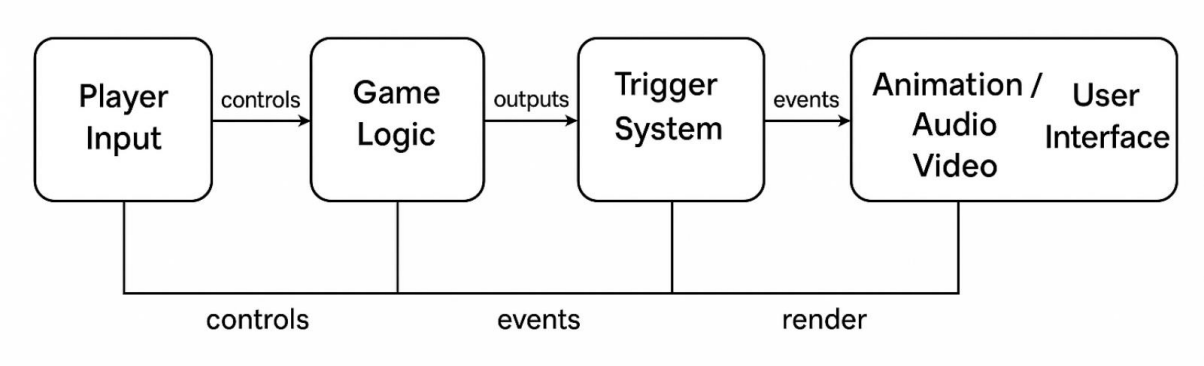


Figure 3.1 System Design Diagram

The architecture illustrates the real-time data flow of the *HUSHHH* game.

1. **Player Input** – Keyboard and mouse signals (movement keys, interaction clicks) form the primary control layer.
2. **Game Logic** – Core C# scripts (player movement, health management, puzzle checks) interpret input, maintain game state, and generate outputs.
3. **Trigger System** – Collider-based events detect proximity or actions (entering zones, pressing keys) and dispatch game events such as monster attacks, countdown starts, or puzzle activations.
4. **Animation / Audio / Video & User Interface** – Final rendering layer where Animator controllers, URP lighting, sound effects, background music, and cut-scene videos are played, while UI elements (health bar, countdown, keypad) update in real time.

Control signals flow left-to-right, while feedback loops (shown as lower arrows) allow continuous updates: player actions directly influence triggers and rendering, ensuring an interactive, immersive gameplay experience.

3.1.1 System Architecture Diagram

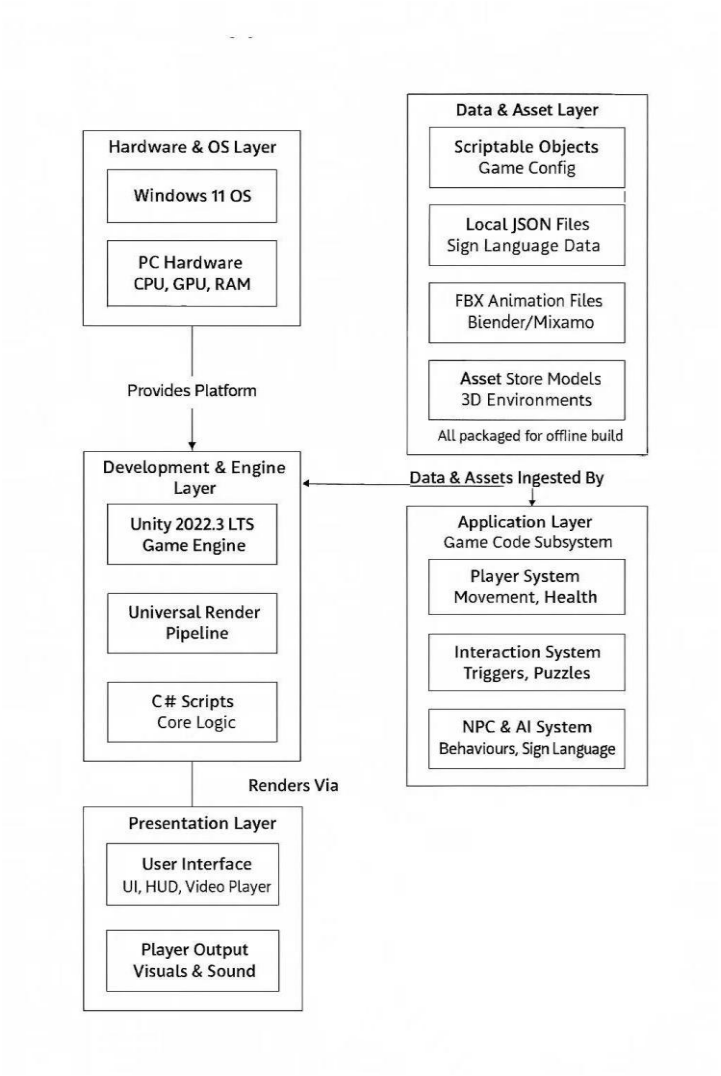


Figure 3.2 System Architecture Diagram

The diagram illustrates the layered architecture of the HUSHHH game, showing how hardware, assets, engine components, and application logic cooperate to deliver real-time gameplay.

Hardware & OS Layer

The foundation of the system consists of the Windows 11 operating system running on consumer-grade PC hardware (CPU, GPU, RAM). This layer provides the computational resources and platform support for Unity and all runtime components.

Data & Asset Layer

Game content originates here. Scriptable Objects store core configuration data; local JSON files hold sign-language reference data; FBX animation files created in Blender (with Mixamo rigs) contain all custom gesture animations; and 3D models from the Unity Asset Store form the environment. These resources are imported by the Unity engine at build time.

Development & Engine Layer

Built with Unity 2022.3 LTS and the Universal Render Pipeline (URP), this layer ingests the data and assets, applying lighting, rendering, and post-processing. Core C# scripts define game logic such as movement, health management, triggers, and puzzle logic.

Application Layer

This layer executes the core game systems:

Player System – Handles movement, health, and input processing.

Interaction System – Manages triggers, puzzles, and sign-language recognition.

NPC & AI System – Controls enemy behaviour, NPC animations, and sign-language demonstrations.

Rendering & Audio System – Combines URP rendering with Unity's AudioSource to produce visuals, music, sound effects, and cut-scenes.

Presentation Layer

The top layer delivers the final experience to the player through the user interface (UI, countdown timer, video player) and real-time audio-visual output. All input feedback loops return here, ensuring responsive controls and immersive interaction.

3.1.2 Use Case Diagram and Description

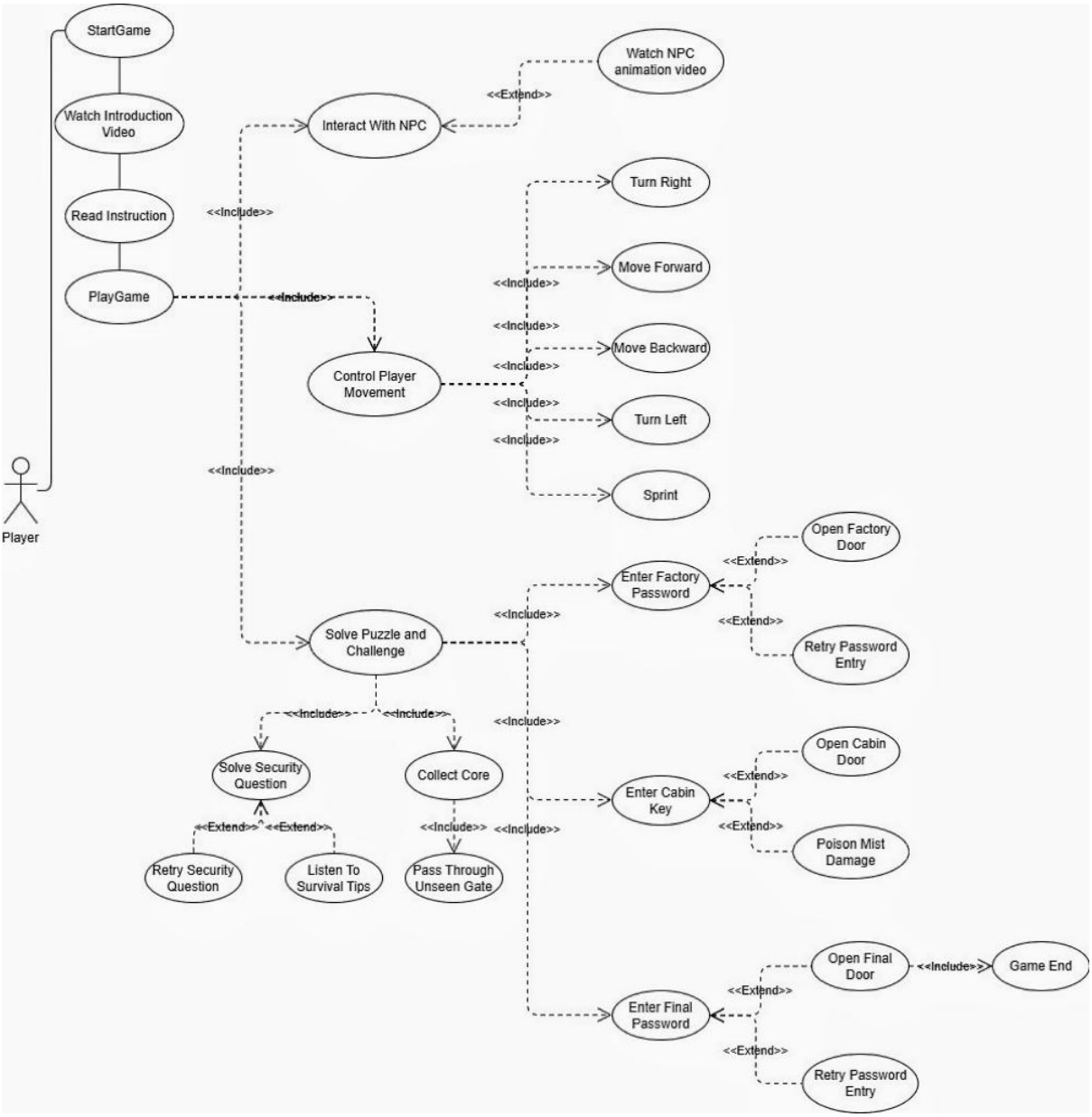


Figure 3.3 Use Case Diagram

The revised *HUSHHH* use case diagram captures the full player journey and all major interactions required to complete the escape-puzzle game. The Player is the only external actor and is responsible for initiating, navigating, and completing every stage of gameplay.

Overall Flow

The experience begins when the player selects Start Game, which leads directly to Watch Introduction Video, a comic-style cutscene explaining the car accident and the mysterious forest setting. After the video, the player proceeds to Read Instruction, where the core

controls—moving forward and backward, turning left and right, and sprinting—are presented. Selecting Play Game then activates the 3D world and hands full control to the player.

Core Gameplay

Once inside the game world, the player can Control Player Movement, which includes moving forward, backward, turning, and sprinting. Movement enables exploration and interaction with all other elements.

The player repeatedly Interacts with NPCs encountered along the path. Entering each NPC's trigger zone initiates animations, and the optional Watch NPC Animation Video extends this interaction by providing sign-language hints.

Puzzle and Challenge Path

The central branch of the diagram, Solve Puzzle and Challenge, groups every objective necessary to escape:

- Collect Core and Pass Through Unseen Gate represent the first critical challenge, requiring exploration to find the hidden core item.
- Enter Cabin Key allows access to the cabin. A correct key press opens the door, while the extension Poison Mist Damage captures the consequence of an incorrect key—health reduction through a poison-mist trap.
- Enter Factory Password grants access to the factory area. If the password is wrong, the Retry Password Entry extension is triggered before the Open Factory Door use case can succeed.
- At the computer terminal, the player must Solve Security Question, which may extend to Retry Security Question on failure or Listen to Survival Tips for hints.
- Finally, Enter Final Password unlocks the ultimate exit. Failure prompts a Retry Password Entry extension, while success leads to Open Final Door and the concluding Game End use case.

3.1.3 Activity Diagram

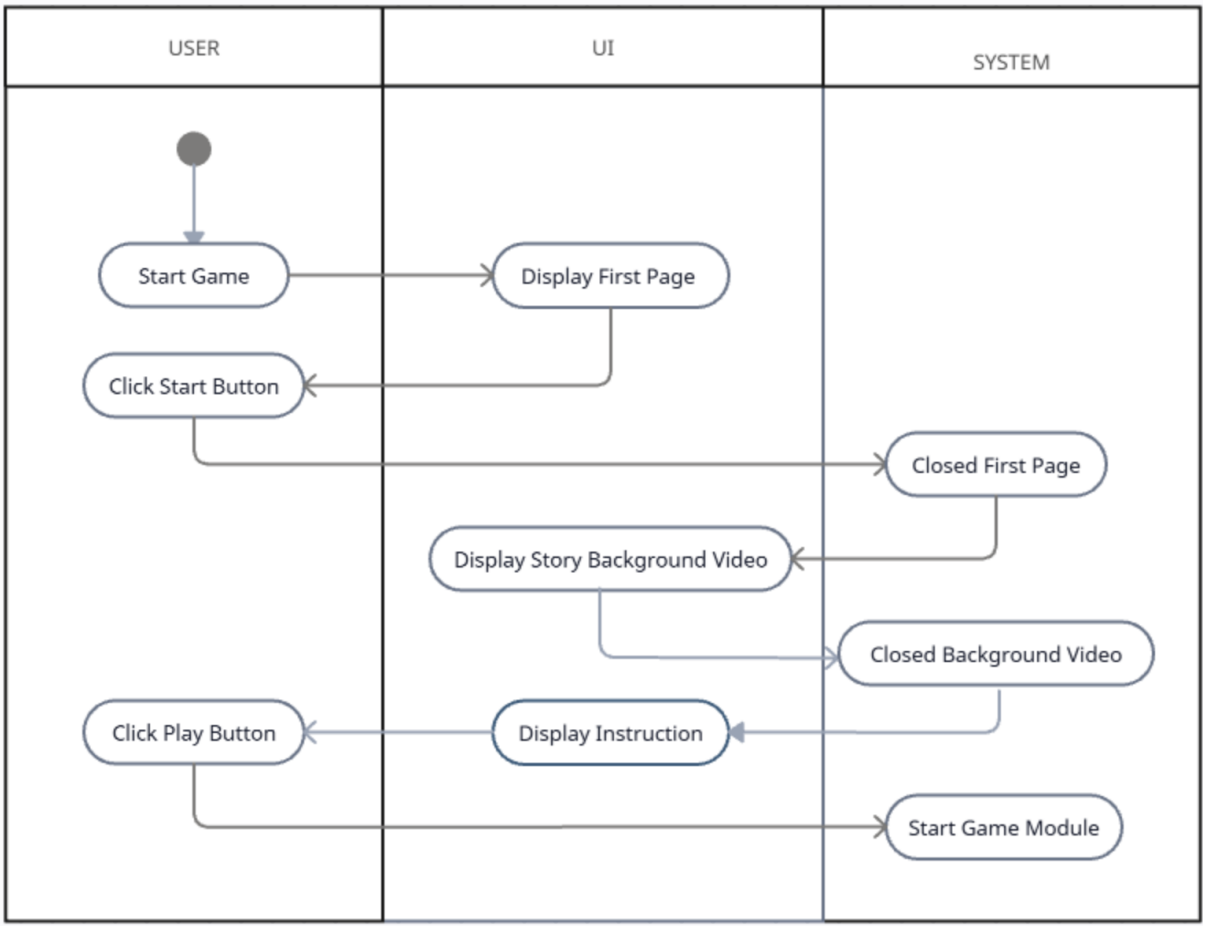


Figure 3.4 Activity Diagram – Game Launch Flow

This activity diagram separate the roles of User, UI, and System, illustrating the complete flow from launching the game to beginning actual gameplay. After the player starts the game, the UI first displays the title screen showing the game name and cover image. When the player clicks the Start button, the system hides the initial UI and the interface plays the story background video. Once the video ends, the system hides the video page and automatically switches to the instruction screen, where key gameplay notes are presented. After the player clicks Continue, the instruction screen closes and the system loads and starts the game module, presenting an interactive 3D environment for exploration. This process clearly demonstrates the sequence and coordination of user actions, interface responses, and system loading, ensuring a smooth transition from launch to full gameplay.

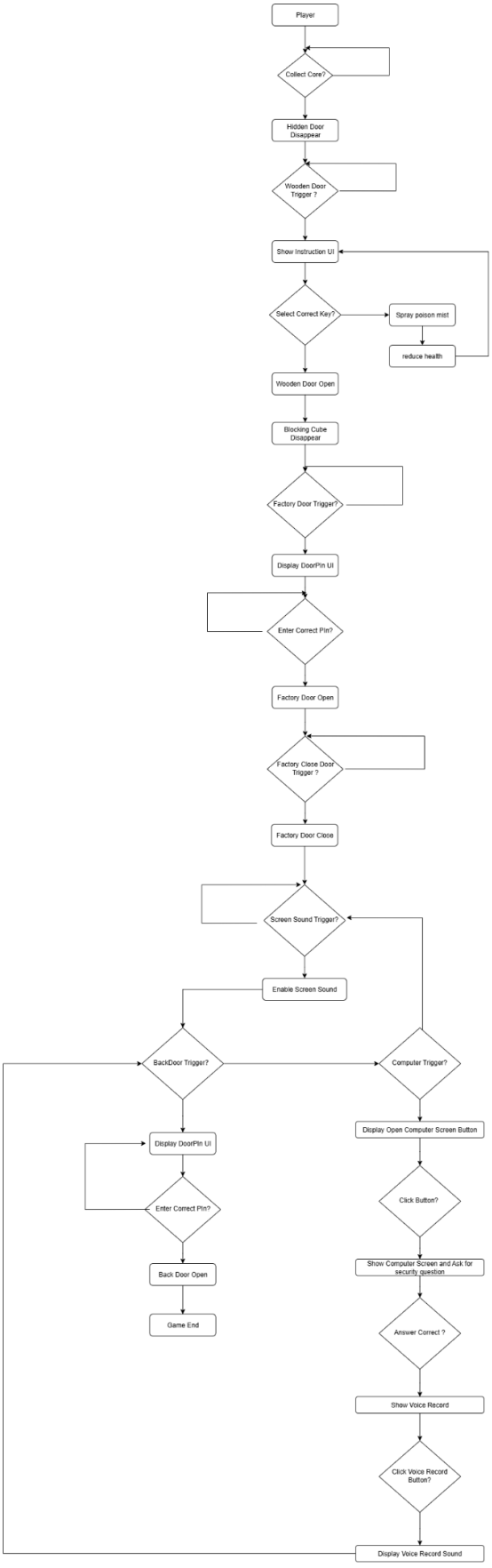


Figure 3.5 Activity Diagram – Gameplay Sequence

This activity diagram outlines the entire gameplay sequence of the sign-language learning escape game, showing how each player action triggers system responses and gradually unlocks the path to final escape.

Initial Stage

The flow begins when the player enters the game world. The first critical decision is to collect the **Core** item. Only after the core is obtained will the hidden entrance become active; if the core is missed, the player must continue exploring until it is found.

Hidden Door and Wooden Door Puzzle

Once the core is collected, the hidden door disappears and the **Wooden Door** trigger becomes active. The system then displays an instruction interface to guide the player in selecting the correct key. A correct selection opens the wooden door, while a wrong choice releases poisonous mist that damages the player's health before allowing another attempt.

Factory Door and Multi-Stage Security

After passing the wooden door, the **Blocking Cube** barrier vanishes, leading the player to the factory door trigger. The system displays a door-pin interface for code entry. Entering the correct pin opens the factory door, which may close automatically once inside. A subsequent screen-sound trigger activates background audio, adding tension and atmosphere as the player proceeds.

Back Door Escape and Computer Interaction

Before reaching the final exit, the player may interact with a computer terminal. Activating the **Computer Trigger** displays an on-screen button to open the computer interface. Selecting this button prompts a security question; only a correct answer reveals a voice-record option. Playing this recording provides the key clue needed to unlock the **Back Door**. With the clue, the player triggers the back-door interface, enters the correct code derived from the recording, and prepares for escape.

Game End

When the correct back-door pin is entered, the back door opens and the game reaches its ending state, marking the player's successful escape.

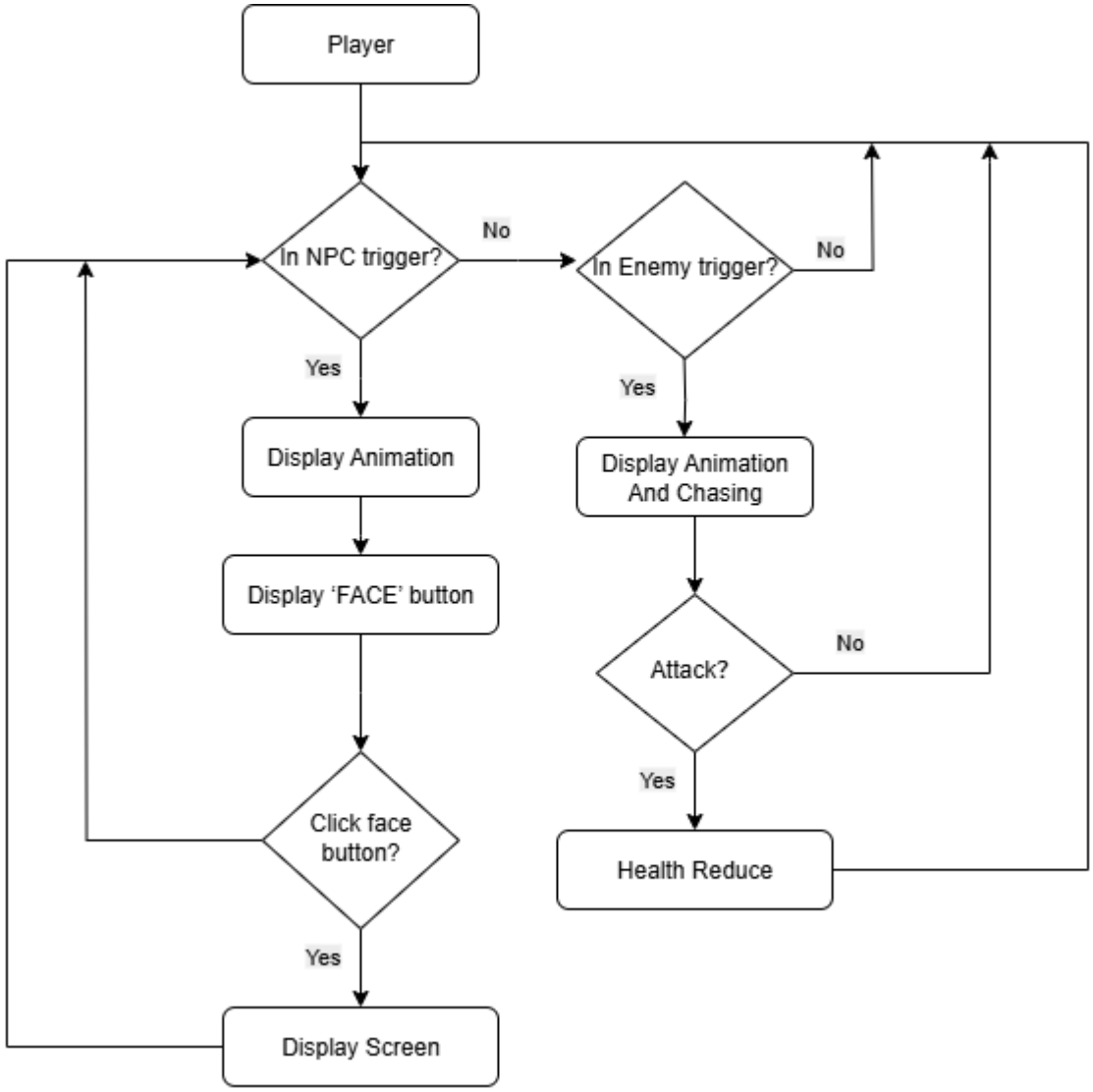


Figure 3.6 NPC and Enemy Interaction Activity Diagram

This activity diagram illustrates the interactive behaviour between the player, nearby NPCs, and hostile enemies during gameplay.

Player Detection and NPC Interaction

The sequence begins as the player moves through the game environment. When the player enters an NPC's trigger zone, the system plays the NPC's idle or greeting animation. A FACE button appears on the user interface, inviting the player to engage. If the player clicks this button, the system presents a full-screen video or cutscene of the NPC, creating the effect of a close-up conversation.

Enemy Trigger and Combat

If the player instead crosses into an enemy trigger zone, the enemy initiates a chase and attack sequence. The system activates the enemy's movement animation and begins pursuit behaviour. A distance check continuously monitors the gap between the enemy and the player. Once the player comes within the defined attack range, the enemy executes an attack animation and the system deducts health from the player. After the attack concludes—or if the player successfully escapes the trigger zone—the flow returns to the starting state, allowing similar encounters to occur again during gameplay.

Chapter 4

System Design

4.1 System Block Diagram

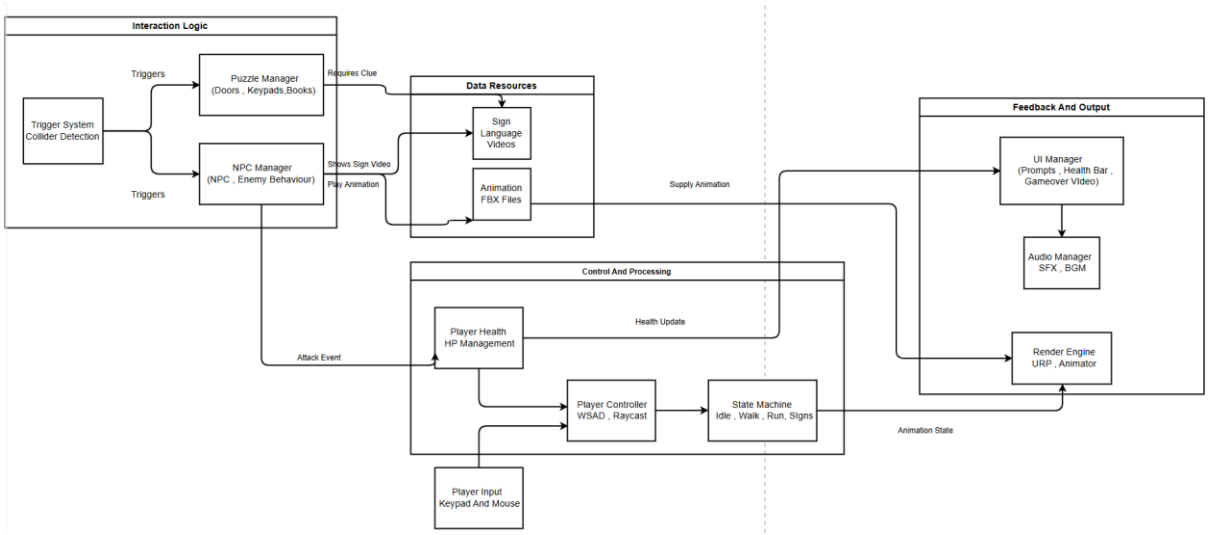


Figure 4.1 System Block Diagram

The diagram illustrates the complete runtime architecture of the HUSHHH game, showing how player input, core logic, data resources, and output feedback interact to deliver real-time gameplay.

1. Interaction Logic

This layer handles every in-game trigger and puzzle event.

Trigger System – Collider Detection monitors the player's position and interactions, dispatching Triggers to downstream modules. Puzzle Manager (Doors, Keypads, Books) manages logic for numeric doors, sign-language locks, and hidden clues, requesting additional data such as sign videos or puzzle status. NPC Manager (NPC & Enemy Behaviour) controls friendly and hostile characters, initiating animations or monster attacks when their trigger conditions are met.

2. Data Resources

Central repositories provide all static content. Sign Language Videos & JSON store gesture references and dialogue cues required for sign-language prompts. Animation FBX Files supply character motions imported from Blender and Mixamo. These resources feed puzzles, NPC behaviours, and UI displays in real time.

3. Control and Processing

This layer maintains the player's state and drives animation logic. Player Input (Keyboard & Mouse) captures movement and interaction commands. Player Controller (WASD, Physics, Raycast) interprets input, applies Unity physics, and communicates with the state machine.

State Machine (Idle, Walk, Run, Signs) updates the current animation state used by the rendering engine. Player Health (HP Management) tracks damage from monster attacks or environmental hazards, sending health updates to the UI.

4. Feedback and Output

The final layer renders visuals, plays sound, and displays UI feedback. UI Manager presents on-screen prompts, health bar, countdown timer, and Game-Over video. Audio Manager handles sound effects and background music. Render Engine (URP, Animator) executes the Universal Render Pipeline to display 3D graphics and apply animation states from the State Machine.

4.2 System Components Specifications

This section describes the major software and logic components of the game, highlighting their responsibilities, key parameters, and implementation details. All components are implemented as C# scripts, Prefabs, and Unity built-in systems within a Unity 2022 URP project.

4.2.1 Player Control Subsystem

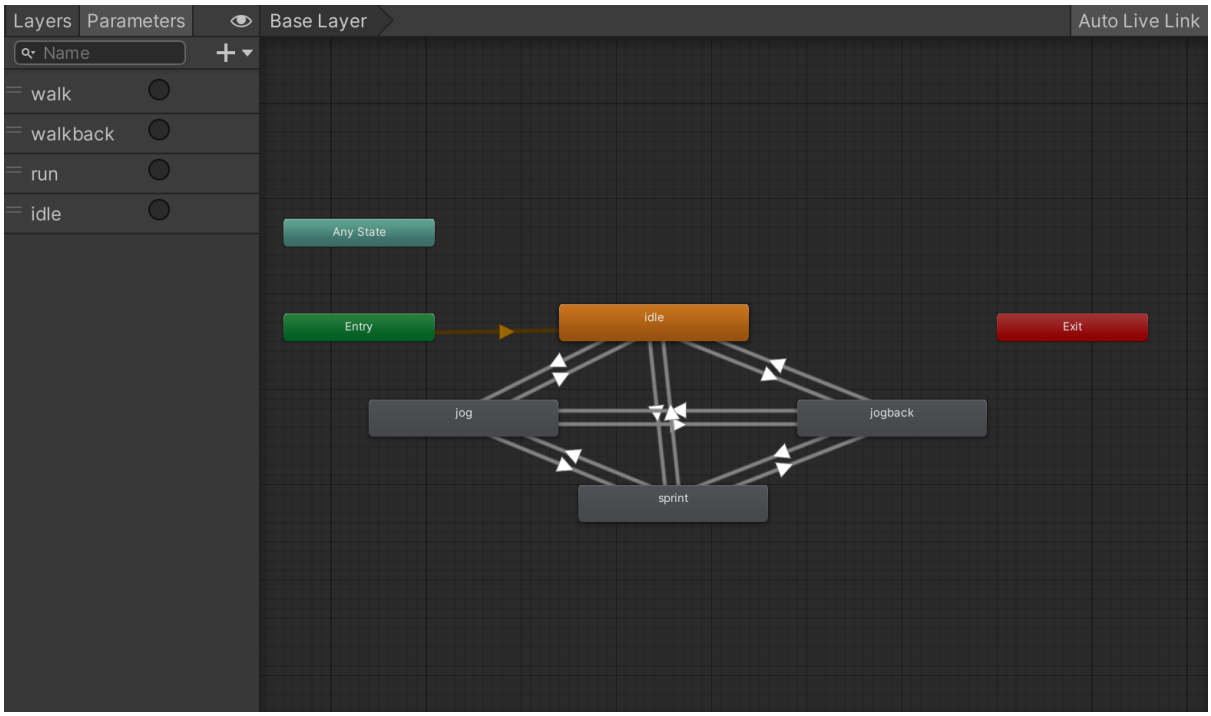


Figure 4.2.1 Player Control State Machine in Unity Animator

Main Script: player.cs

Functions: Captures W/A/S/D and Shift input to handle forward/backward motion, smooth rotation, and sprinting.Uses Rigidbody physics and raycasts to detect ground slope and step height, adding assisting force for natural climbing. Passes movement states to the Animator Controller to switch between Idle, Walk, WalkBack, and Run.Key Parameters: walking/running speeds, maximum slope angle, step-height threshold.

4.2.2 Player Health & Game State

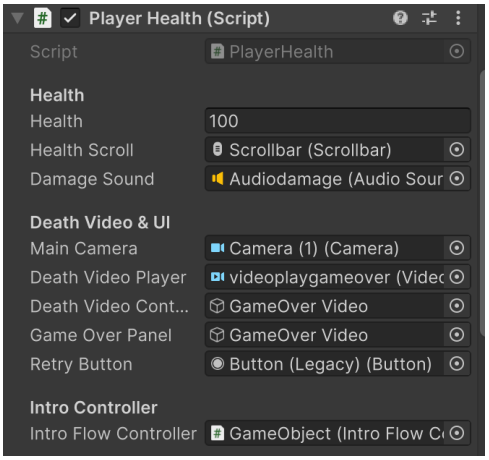


Figure 4.2.2 Player Health & Game State – Unity Inspector View

Main Script: PlayerHealth.cs

Functions: Maintains health values and updates the on-screen Scrollbar. Plays hit sound and deducts health on TakeDamage. When health reaches zero, disables player control and camera, plays the Game-Over video, and displays the retry panel. Exposes ForceGameOver() so external scripts (e.g., countdown) can trigger the same sequence.

4.2.3 Enemy & NPC Subsystem

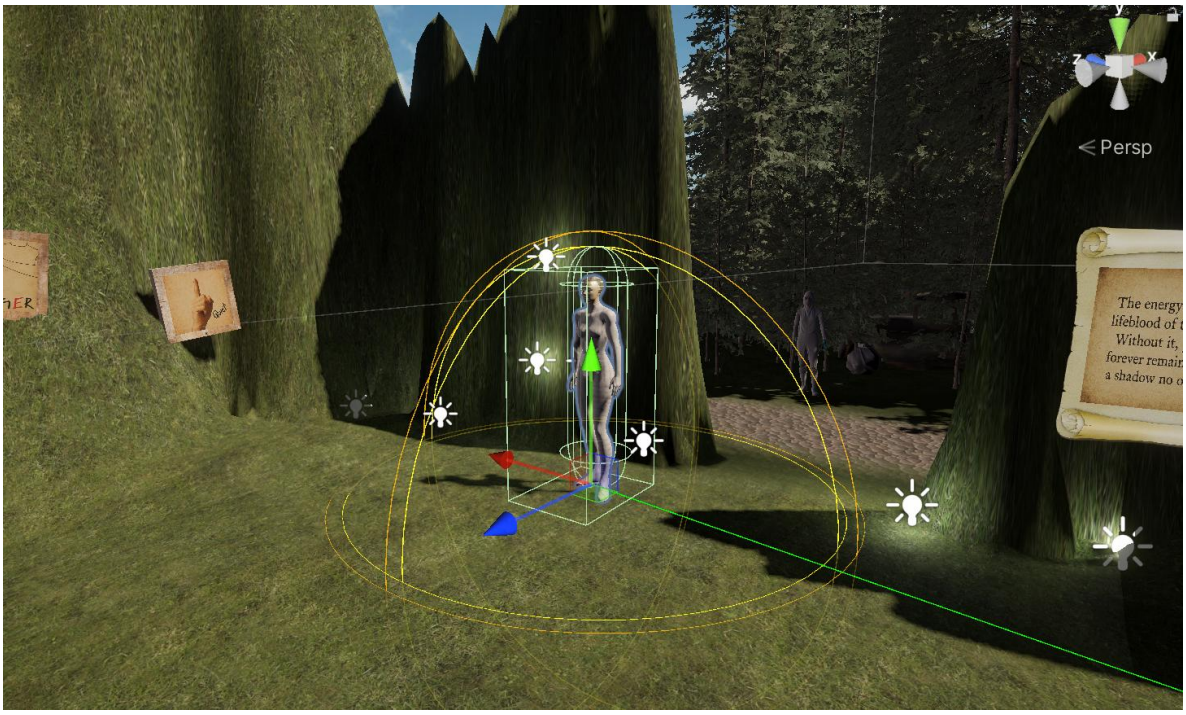


Figure 4.2.3 Enemy & NPC Subsystem – NavMesh and Detection Setup

Scripts: MyEnemy.cs, dangerScript.cs

Functions: Uses NavMeshAgent to patrol and chase the player, switching Animator states between Idle, Run, and Attack based on distance. NPC scripts detect proximity, face the player, and trigger alert animations before resetting to Idle.

4.2.4 Environment Triggers & Interaction

Figure 4.2.4 Environment Trigger & Interaction – Collider Zones

Representative Scripts:

CountdownGameOverTrigger – Starts a 10-minute countdown when the player enters a zone; calls ForceGameOver() at zero.

PressKeyOpenDoor / PressKeyOpenDoorWithMist – Opens doors on key press, optionally spawns poison mist, plays sounds, and damages the player for incorrect input.

ZoneUIController – Handles a multi-step computer interface, including a "deduct time" button that reduces the countdown and plays a one-shot sound effect.

KeypadTrigger – Displays a keypad panel, unlocks the cursor for password entry, and restores its state when the player exits.

4.2.5 User Interface (UI) Subsystem

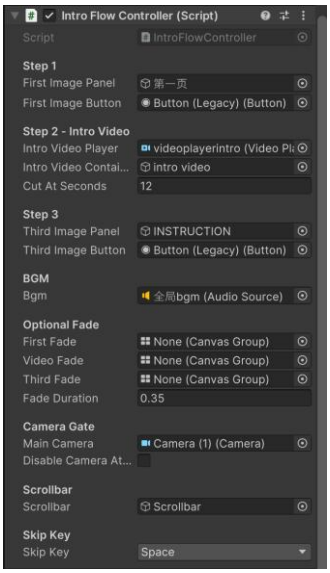


Figure 4.2.5 User Interface Subsystem – Unity Inspector View

Technology: UnityEngine.UI (Button, Text, Scrollbar, DoorPin, CanvasGroup)

Functions: IntroFlowController manages the intro sequence (image → video → final panel) with background music and fade effects. Countdown panels, prompts, and retry buttons use Button.onClick.AddListener for event binding. CanvasGroup and coroutines provide smooth fade-in/fade-out transitions.

4.2.6 Audio & Video Playback

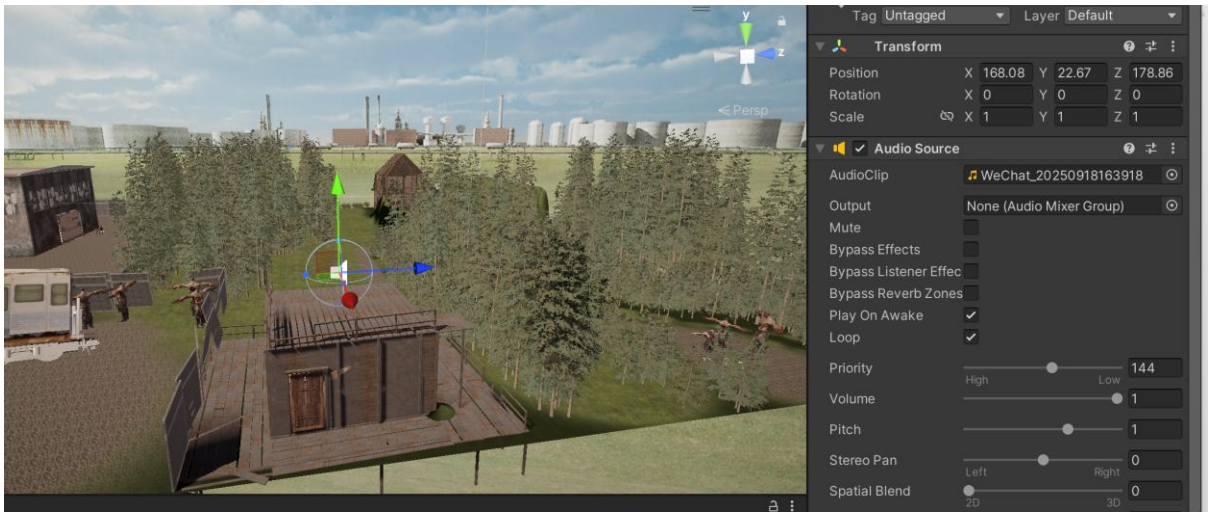


Figure 4.2.6 Audio & Video Playback

Components: AudioSource, VideoPlayer

Functions: Plays environment sounds, UI clicks, damage effects, and "deduct time" audio.

Handles intro, death, and victory cut-scenes with Prepare(), loopPointReached, and errorReceived callbacks. Releases RenderTexture after playback to prevent frame ghosting.

4.2.7 Data & Resource Management

Storage: Fully local; no external database or network connection required.

Organization: Unity Scenes, Prefabs, Scriptable Objects, and serialized fields store all parameters (trigger ranges, countdown values, video paths) and are packaged inside the build for offline play.

4.3 Circuits and Components Design

4.3.1 Introductory User Flow and Instruction Delivery

Figure 4.3.1 Introductory User Flow

The game opens on a main menu screen featuring the bold, blood-red title HUSHHH, immediately setting a dark, unsettling tone. A vintage-style illustration of an abandoned industrial district appears at center, with a single PLAY button inviting the player to begin. Clicking PLAY triggers a comic-style cut-scene with sound, which doubles as a narrative prologue and the first tutorial cue. In this illustrated sequence, the protagonist drives along a forest road near a sprawling refinery. A sudden dizziness causes her to lose control, and the car crashes violently into a tree—CRASH! before she awakens in a silent, eerie forest surrounded by mysterious picture frames. This cinematic introduces the backstory and highlights the importance of the sign-bearing frames, guiding players toward the core mechanics and narrative objective.

4.3.2 Environment and Scene Design

Figure 4.3.2 Environment and Scene Design – Industrial Outdoor Terrain

The game world is built around a large, explorable outdoor terrain and extended with a high-resolution **Skybox** to create the illusion of endless space. Industrial-themed panoramic textures in the skybox depict distant chemical plants, storage tanks, and pipelines, giving the entire map the atmosphere of an abandoned industrial zone. Although the player's actual movement area is limited, this layered background treatment generates a convincing sense of vastness and echoes the narrative of a failed research facility.

4.3.3 Abandoned Boundaries and Natural Terrain

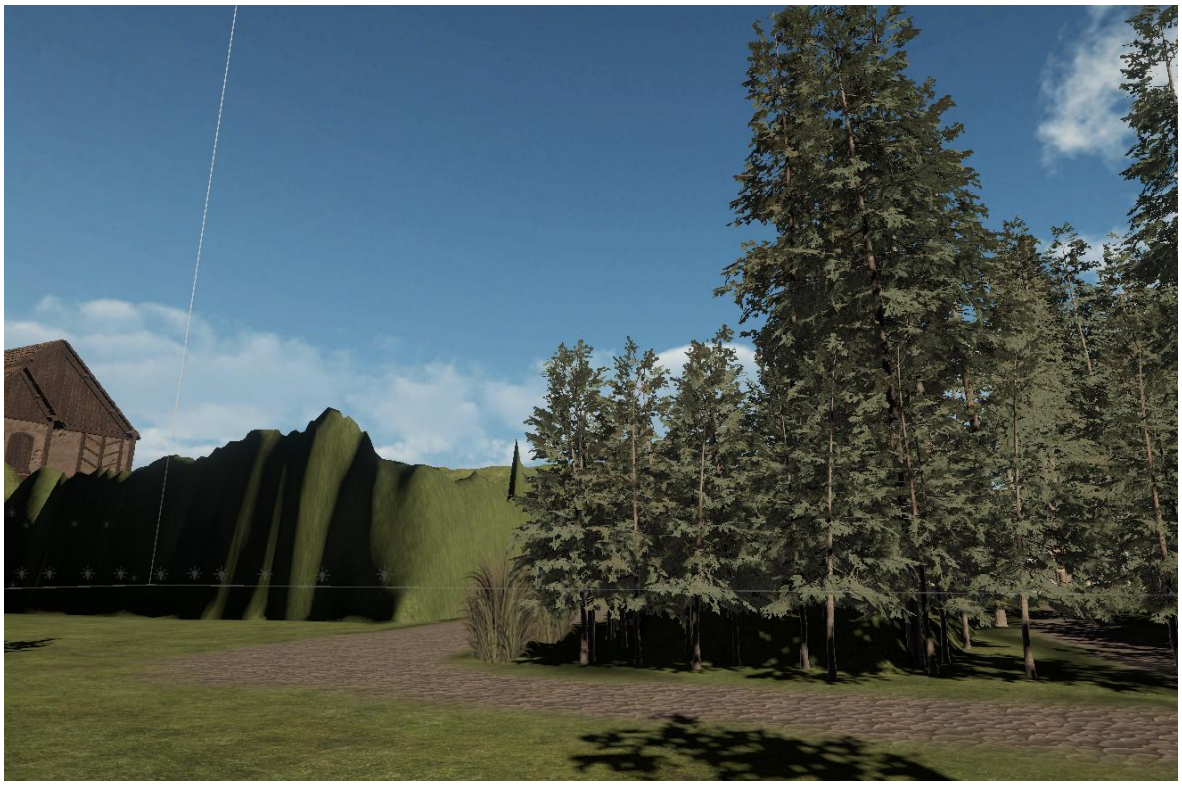


Figure 4.3.3 Abandoned Boundaries and Natural Terrain – Forest Area

Dynamic lighting and **subtle fog effects** were designed to create areas of concealment and tension, giving the impression of a changing atmosphere as the player explores. The world itself is a large, explorable outdoor terrain extended with a high-resolution **Skybox** to create the illusion of endless space. Panoramic industrial textures depict distant chemical plants and storage tanks, reinforcing the abandoned-facility theme. Although the player's actual movement range is limited, layered backgrounds generate a convincing sense of vastness.

4.3.4 Key Structures and Interior Layouts

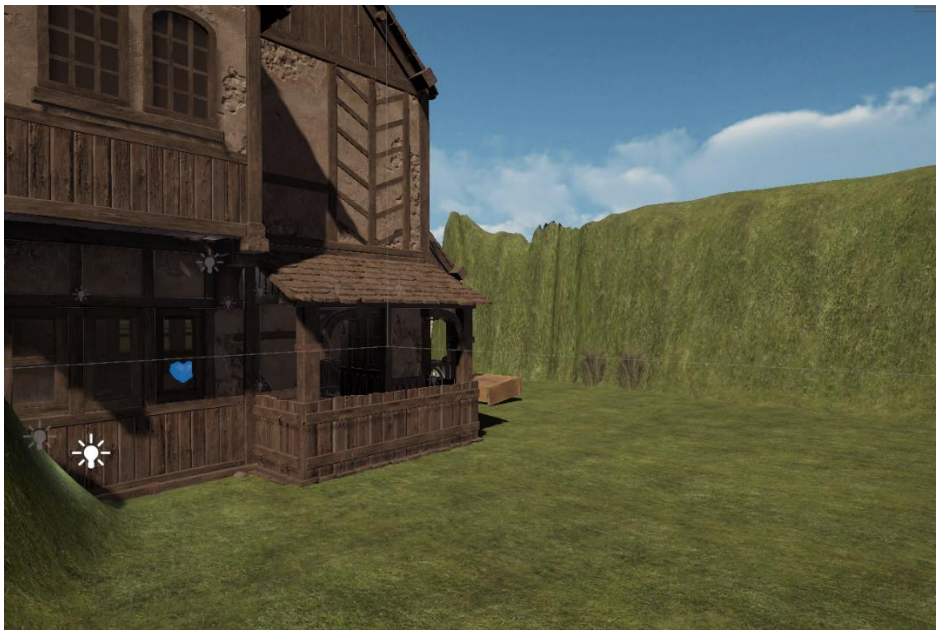


Figure 4.3.4 Key Structures and Interior Layouts – Village Building

Figure 4.3.5 Key Structures and Interior Layouts – Office

Deeper in the forest stand a weathered **wooden house** and a small **office complex**, remnants of the former research site. Peeling timber surfaces, broken window frames, and toppled furniture create the impression of long abandonment. Inside, desks, filing cabinets, and computer terminals host puzzles and story triggers. Dynamic lighting and fog intensify as the player approaches, hiding clues while heightening suspense.

4.3.5 Monsters and Horror Atmosphere

Figure 4.3.6 Monsters and Horror Atmosphere – Monster In Forest

Grotesque creatures roam both the fenced compound and forest trails. Detailed normal-mapped textures rendered in real time by the **Universal Render Pipeline (URP)** create a visceral, distorted appearance. AI scripts handle patrol, detection, pursuit, and attack behaviours, so that when the player enters a trigger zone the monsters react instantly, amplifying the horror experience.

4.3.6 Sign Language Recall

Figure 4.3.7 Hazard Sign Recall and Laboratory Entrance

This screenshot captures the final stage of the level where the player approaches the hidden laboratory door. A previously introduced danger sign-language gesture reappears as a critical clue, prompting the player to recall and replicate the correct sign. Correct recognition allows safe entry, while failure to remember the gesture triggers the next hazard event.

4.3.7 Final Scene

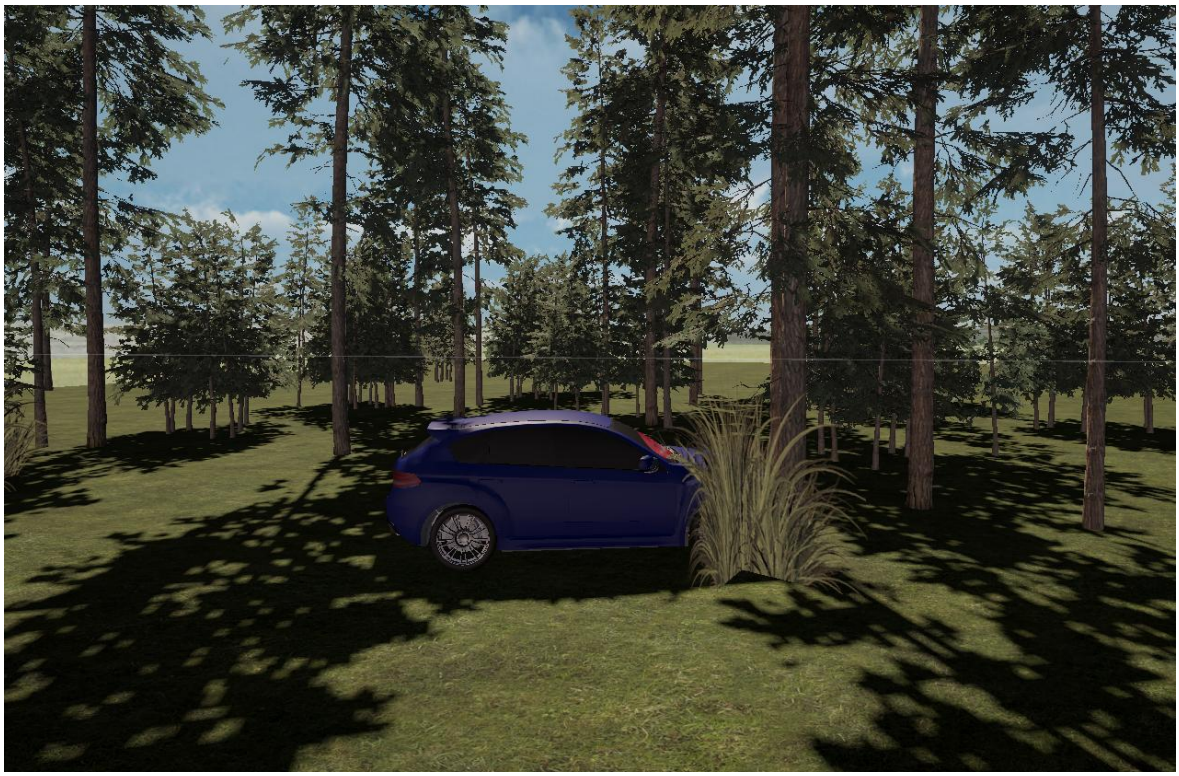


Figure 4.3.8 Narrative Echo and Ending Scene – Final Escape with Car

The final escape deliberately echoes the opening comic. After solving puzzles and surviving monsters, the player discovers a **blue car** parked at the forest's edge—the same vehicle from the prologue. Careful lighting and camera framing make the closing scene resonate with the introduction, signalling a full narrative circle and a satisfying conclusion.

4.3.8 Animation and Rendering

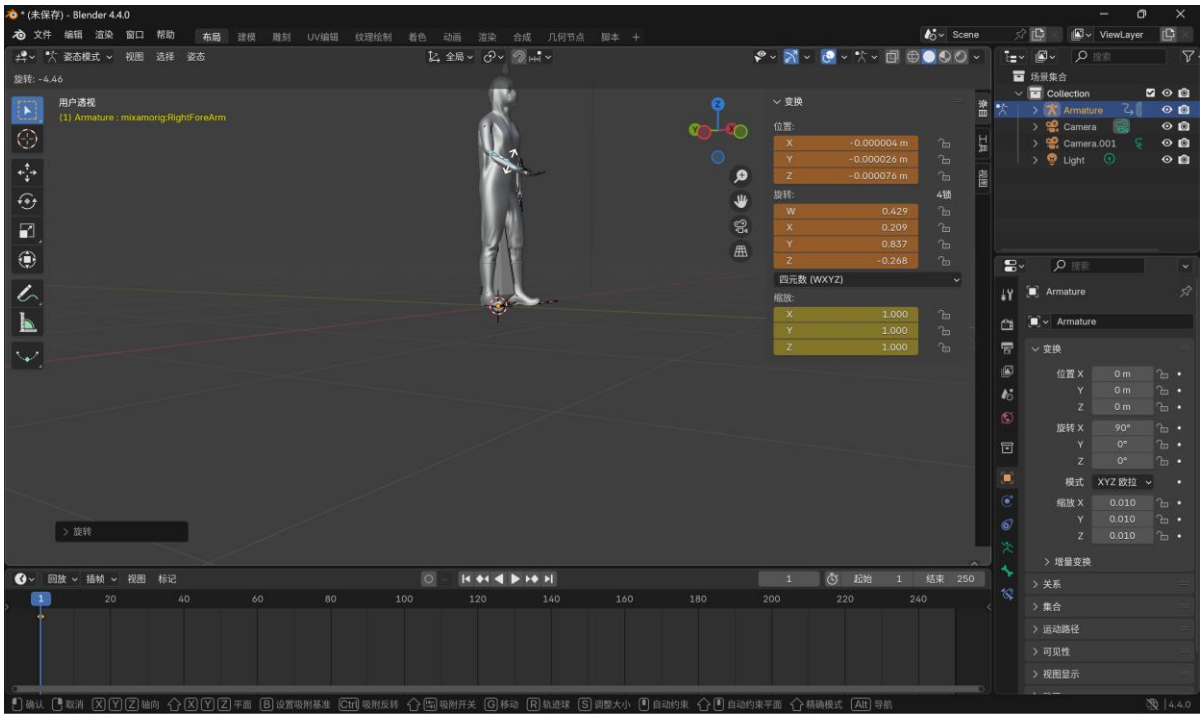


Figure 4.3.9 Animation and Rendering – Blender Sign-Language Rig and Export

All sign-language instructional animations and several character actions were first created frame by frame in Blender and exported as FBX files. These animation assets were then imported into **Unity** and connected to **Animator Controllers**, which receive parameters from gameplay scripts to trigger smooth transitions between states such as *Idle*, *Walking*, *Running*, *Sign-Language Gestures*, and *Enemy Attacks*. During runtime, the **Universal Render Pipeline (URP)** handles real-time lighting, shadowing, and post-processing effects, ensuring that every animation—from player movement to NPC gestures and enemy attacks—appears consistent and immersive throughout the game world.

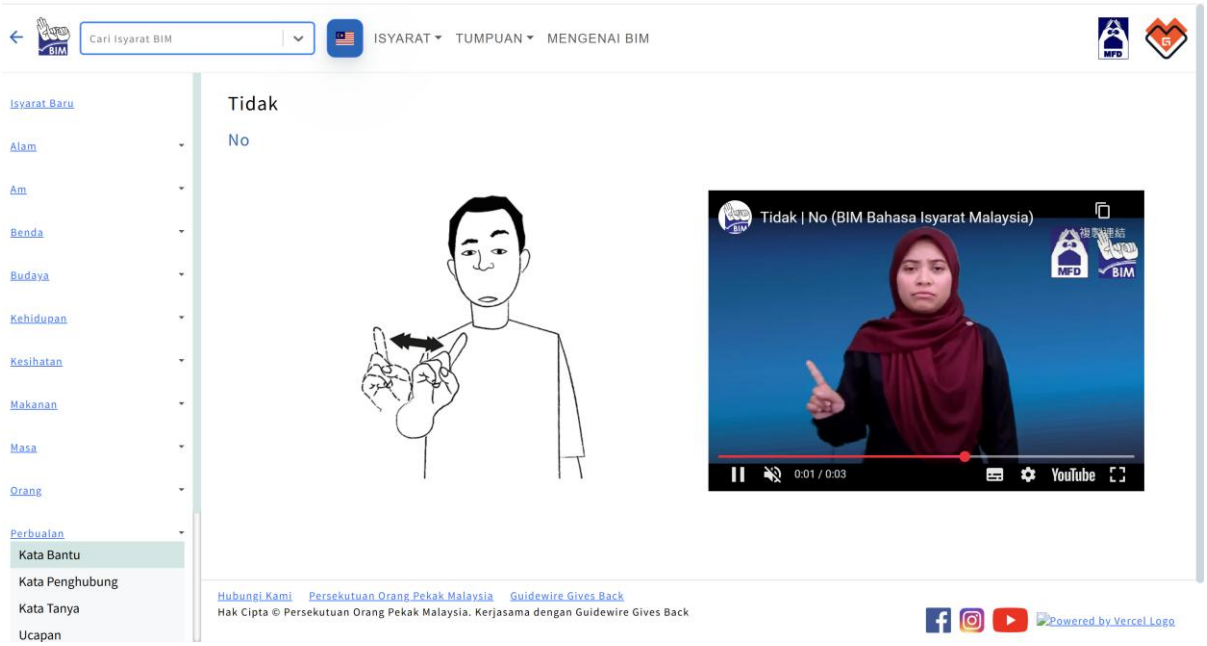


Figure 4.3.10 Malaysia Sign Bank Reference – BIM Gesture Resource Screenshot

The authenticity and linguistic accuracy of the sign language interactions presented in this project are fundamentally reliant on the comprehensive resources provided by the **Malaysia Sign Bank** (BIM Sign Bank). As the most authoritative and extensive digital repository dedicated to **Malaysian Sign Language (BIM)**, it served as the definitive reference for animating all NPC communications. This ensured that every gesture, from basic greetings to complex narrative-driven phrases, was meticulously crafted to reflect genuine BIM usage, thereby preserving the cultural and communicative integrity essential for an educational game developed within the Malaysian context.

4.3.9 Sign-Language Clue Boards

Figure 4.3.11 Sign-Language Clue Boards – Outdoor Placement

Figure 4.3.12 Sign-Language Clue Boards –Indoor Placement

To assist players who are not familiar with sign language, the environment includes a series of weathered clue boards placed strategically along key paths and inside buildings.

- Each board displays an illustrated hand sign with a short annotation (e.g., numbers or simple English words such as *Green*, *Across the Street*).
- These hints help players interpret the gestures performed by NPCs and understand the actions required to progress through puzzles.
- The boards are textured to look aged and slightly damaged, blending naturally with the abandoned-factory aesthetic while providing clear instructional value.

4.4 System Components Interaction Operations

This section explains the complete runtime interactions among the major components of the game, showing how player input, control scripts, triggers, events, animation, sound, and UI updates form a continuous processing chain:

Player Input → Control Scripts & State Machine → Trigger Detection → Event Handling → Animation & Audio Rendering → UI Feedback.

4.4.1 Player Input and Character Control

The player controller captures keyboard commands (W/A/S/D and Shift) and updates the movement state each frame. In FixedUpdate, it applies velocity to the Rigidbody for physical motion while checking ground slope and step height via raycasts. When climbing a slope or step, an assisting force ensures smooth movement. State changes feed into the Animator Controller, transitioning between Idle, Walk, WalkBack, and Run.Health is managed by PlayerHealth, which updates the on-screen health bar and plays damage sounds when TakeDamage() is called. If health reaches zero, player control scripts are disabled and a Game-Over video is triggered.

4.4.2 Environment Triggers and Interactive Events

Unity's **Collider** and *isTrigger* system detects when the player enters special zones and invokes scripted logic:

- MyEnemy checks the distance to the player and its start position to decide whether to patrol or chase; within attack range it triggers an attack animation and calls PlayerHealth.TakeDamage().
- dangerScript rotates NPCs toward the player and plays an alert animation when approached.
- Puzzle mechanisms such as PressKeyOpenDoor, ShowCharacterOnTrigger, and ZoneUIController display UI panels, play animations, or start audio when the player interacts, and reset when the player leaves.
- KeypadTrigger activates a password keypad UI, unlocks the mouse cursor for input, and restores the previous cursor state on exit.

4.4.3 Animation and Rendering

All sign-language instructional animations and character motions were created in **Blender** and exported as FBX files. Inside Unity, **Animator Controllers** receive parameters from scripts to drive smooth transitions—idle, jogging, sign gestures, or enemy attacks. The final visuals are rendered in real time by **URP**, ensuring consistent lighting and post-processing effects.

4.4.4 Audio and UI Feedback

Sound and UI updates are tightly bound to gameplay events.

- PlayerHealth adjusts the health bar and plays hit sounds on damage.
- ZoneUIController manages a multi-step computer interaction, activating buttons, revealing images, confirming steps, and triggering audio.
- VideoAudioTrigger unmutes a video when the player enters its trigger zone and mutes it again on exit.
- IntroFlowController coordinates the splash screen, story video playback, fade transitions, background music, and final activation of the main camera.

4.4.5 Data Flow Between Modules

Player inputs are processed by the **player script**, which drives physics and animation states. Trigger events send positional and state information to **enemy AI**, **NPC scripts**, and **environment controllers**. These, in turn, pass events to the **Animator**, **AudioSource**, and **UI Manager** to produce visual/audio feedback or to PlayerHealth to apply damage. The keypad scripts communicate with UI elements and security checks to confirm code entry. Through this tightly coupled data exchange, the game maintains a **real-time feedback loop** that supports both the tense horror atmosphere and the precise sign-language teaching objectives

Chapter 5

System Implementation

5.1 Hardware Setup

Specification	Details
Model	ASUS Vivobook OLED 15
Processor	Intel® Core™ i7-12700H, 14 cores (6 Performance + 8 Efficiency), base 2.3 GHz, turbo up to 4.7 GHz
Memory	16 GB DDR4 RAM
Storage	512 GB PCIe NVMe Solid-State Drive
Display	15.6-inch OLED Full HD (1920 × 1080)
Graphics	Integrated Intel® Iris Xe and discrete NVIDIA® GeForce RTX 3050 with 4 GB GDDR6
Operating System	Microsoft Windows 11 Home, 64-bit

Table 5.1 Hardware Setup Specification

5.2 Software Setup

Function	Tool/Technology	Simple Explanation
Game Engine	Unity	The main platform for game development, offering terrain generation, environmental rendering, etc.
Terrain Generation	Unity Terrain	Uses Unity's Terrain tools to create natural terrain in the game world.
Character Modeling & Animation	Blender, Mixamo	Blender is used to create custom sign language animations, while Mixamo provides animation resources.
Sign Language Input Management	C# (SignLanguageInteractionManager)	Develops C# scripts to manage player sign language input and respond to game logic.
Animation System	C# (NPCAnimationSystem)	Uses C# scripts to control NPC (non-player character) animations.

Table 5.2 Software Setup Specification

Technical Development and Implementation Strategy The project will leverage Unity as the primary game development engine, utilizing an ASUS VivoBook 15 OLED for development. The workflow will involve terrain generation using Unity's Terrain tools, detailed environmental rendering, and asset acquisition from Unity Asset Store and Mixamo. Character models, animations, and environmental elements will be sourced and customized, with specialized sign language animations created in Blender. C# scripts will be developed to implement core game mechanisms, including a SignLanguageInteractionManager class for managing player sign language input and system responses, ensuring precise execution of interaction logic within the Unity environment.

Modular Development and Extensibility Approach To ensure code maintainability and future scalability, the project will adopt object-oriented design principles with decoupled functional modules. Key C# script modules will include PlayerController and others, each adhering to the single responsibility principle. By utilizing Unity's MonoBehaviour base class and interface design, the development will achieve loose coupling, allowing for seamless integration of sign language mechanics and creating a flexible framework that supports the game's educational objectives of introducing sign language through an immersive survival game experience.

5.3 Setting and Configuration

Development Environment Configuration

- **Unity Project Settings** – Unity 2022 URP pipeline enabled for real-time lighting, post-processing, and high-quality rendering. Project set to DirectX 11 with linear colour space and a default resolution of 1920 × 1080.
- **Version Control** – Source control maintained through a private GitHub repository, allowing incremental commits and rollbacks. *(Optional: Git LFS enabled for large binary assets such as textures and FBX models.)*
- **Build Settings** – Target platform: Windows 64-bit, compression method LZ4HC for faster loading. All gameplay scenes (Intro, Outdoor Map, Factory Interior, Ending) included in the build index in proper sequence.
- **Player Settings** – V-Sync disabled and target frame rate capped at 90 FPS to balance GPU load and battery usage. Full-screen windowed mode selected to support multiple monitor sizes.
- **External Tools** – Visual Studio 2022 for C# script editing with Unity plug-in integration; Blender for hand-crafted sign-language animations exported as FBX; Mixamo used for base humanoid character rigs prior to Blender refinement.

Game-Side Configuration

- **Input Mapping** – W/A/S/D + Shift for movement and sprint; mouse for camera control; custom key bindings stored in Unity's Input Manager for easy rebinding in future versions.
- **UI** – Scrollbar (health), countdown timer, and interactive prompts organized in separate Unity canvases to simplify activation/deactivation during cut-scenes and end-game events.
- **Audio/Video** – Audio Mixer groups defined for background music, SFX, and video playback to enable independent volume control. All cinematic videos (intro, victory, game-over) pre-loaded and streamed from the build to avoid runtime buffering.
- **Quality Profiles** – Two quality levels ("High" and "Standard") provided so lower-spec machines can reduce shadow resolution and anti-aliasing without altering gameplay logic.

Deployment Configuration

The final executable is a stand-alone Windows build distributed as a single folder containing the Unity Player, required DLLs, and a "Videos" subfolder for cinematic assets. No external database or internet connection is required; all textures, animations, and audio files are packaged within the build for fully offline play, ensuring consistent behavior across different test machines.

5.4 System Operation

5.4.1 Intro / Main Menu & Prologue

Figure 5.4.1 HUSHHH Main Menu Screen with "Play" Button

Figure 5.4.2 Background Story - Driving and Crash Sequence

Figure 5.4.3 In-Game Instruction Interface with Survival Tips

When the game starts, the player first clicks the Play button on the main menu. The system then automatically plays a cinematic video with background audio to introduce the background story. After the video finishes, the screen smoothly transitions to the Instruction interface, where the core controls and key gameplay notes are presented.

Once the player has read the instructions and clicks the Start button, the game officially enters the interactive phase: the main camera and control scripts are activated, allowing the player to freely explore and interact within the 3D environment

5.4.2 NPC Interaction & Sign-Language Prompt

Figure 5.4.4 NPC Triggered Animation with On-Screen Interaction Button

Figure 5.4.5 Player Engaging with NPC Using Sign-Language Gesture

As shown in the figure, when the player enters the trigger zone, the NPC automatically performs the corresponding animation and a Button appears on the interface. By clicking this button, the player can view a close-up presentation of the NPC's sign-language actions, creating the effect of a direct, face-to-face interaction.

5.4.3 Core Collection & Indoor Exploration

Figure 5.4.6 Outdoor Maze and Core Collection Sequence

Figure 5.4.7 Indoor Core Item - Collectible Heart Model

The player cannot pass through the invisible door directly. To gain access, they must first follow the hints posted on the walls and retrieve the core item. Only after obtaining this core will the system recognize the player as a local resident, allowing the hidden door to open.

5.4.4 Monster Chase & Combat

Figure 5.4.8 Player Approaching Friendly NPC for Guidance

Figure 5.4.9 Monster Attack Event During Chase Sequence

If the player fails to recognize the hand-sign that signals danger, they may unknowingly enter the monster's patrol zone. Once inside this area, the monster's detection system activates, triggering an immediate chase and attack sequence. During the pursuit, every successful strike from the monster reduces the player's health, creating intense pressure and reinforcing the importance of correctly understanding the warning sign.

5.4.5 Password Door Puzzle

Figure 5.4.10 Digital Door Interface Prompt for Password Entry

When the player approaches the door, an on-screen prompt appears. To unlock it, the player must interpret the surrounding clues and deduce the correct button/input sequence to activate the door mechanism.

Figure 5.4.11 Correct Password Unlocking the Factory Door

If the player presses an incorrect button, the door emits a burst of toxic gas, immediately reducing the player's health. Entering the correct sequence, however, safely unlocks and opens the door, allowing the player to proceed.

5.4.5 Hidden Clues

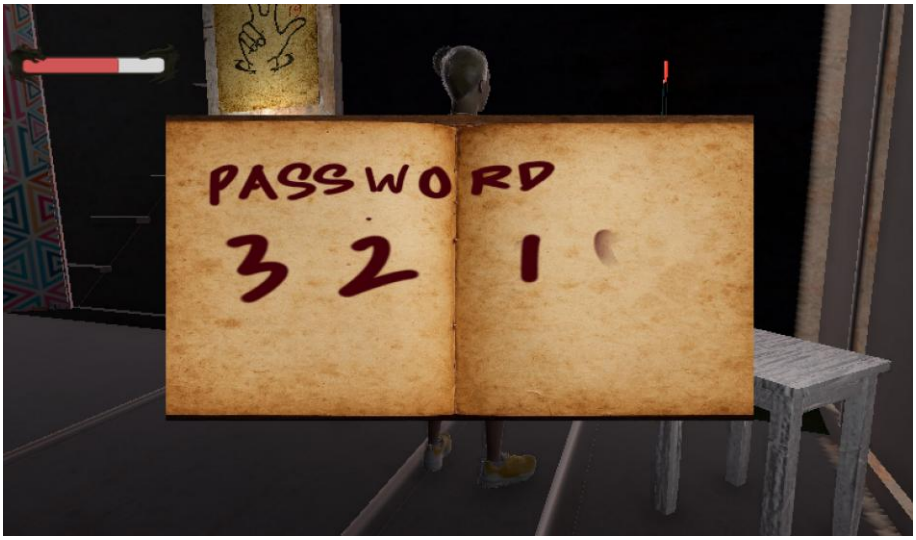


Figure 5.4.12 Open Book Puzzle Revealing Warehouse Code

When the player enters the room and approaches the book, an **Open Book** UI prompt appears on the screen. Selecting this prompt reveals the hidden warehouse door code, enabling the player to proceed to the storage area and unlock the next stage of the game.

5.4.6 Forest Path

Figure 5.4.13 Forest Path with Elf Guide

Figure 5.4.14 Enemy Encounter

When the player comes to a fork in the road, an elf will appear to guide the player to the right path. If the player chooses the wrong path, he will end up in a group of zombies.

5.4.7 Warehouse & Security Question

Figure 5.4.15 Warehouse Door Pin

Figure 5.4.16 Warehouse Door Open

When the player enters the trigger range of the warehouse door, a digital password will appear. The player can delete the number entered by clicking the delete button in the lower right corner and submit the password by clicking the submit button in the lower left corner. After successful submission, the door will automatically open with a heavy door opening sound.

Figure 5.4.17 Warehouse Door Auto Close

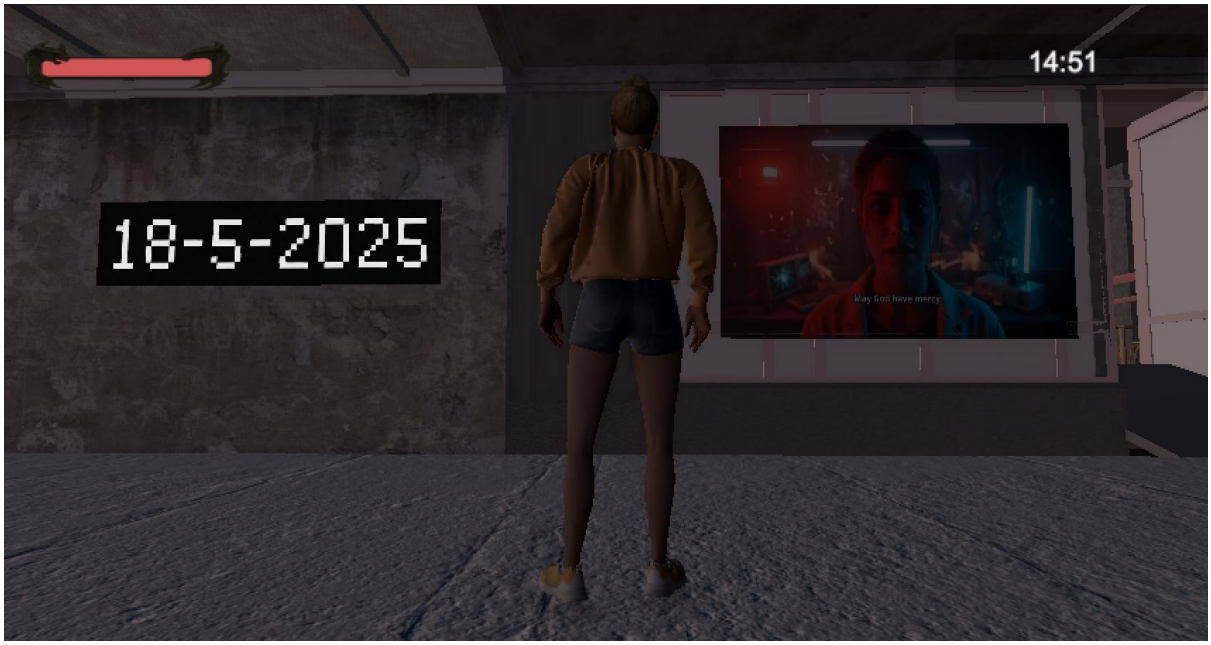


Figure 5.4.18 Screen Video And Countdown Time Trggered

After the player enters the trigger zone, the door automatically closes behind them, trapping the player inside and forcing them to complete the final level. Once inside, a large screen activates and plays a video announcement. At this moment, a 15-minute countdown begins, signaling the start of the final challenge.

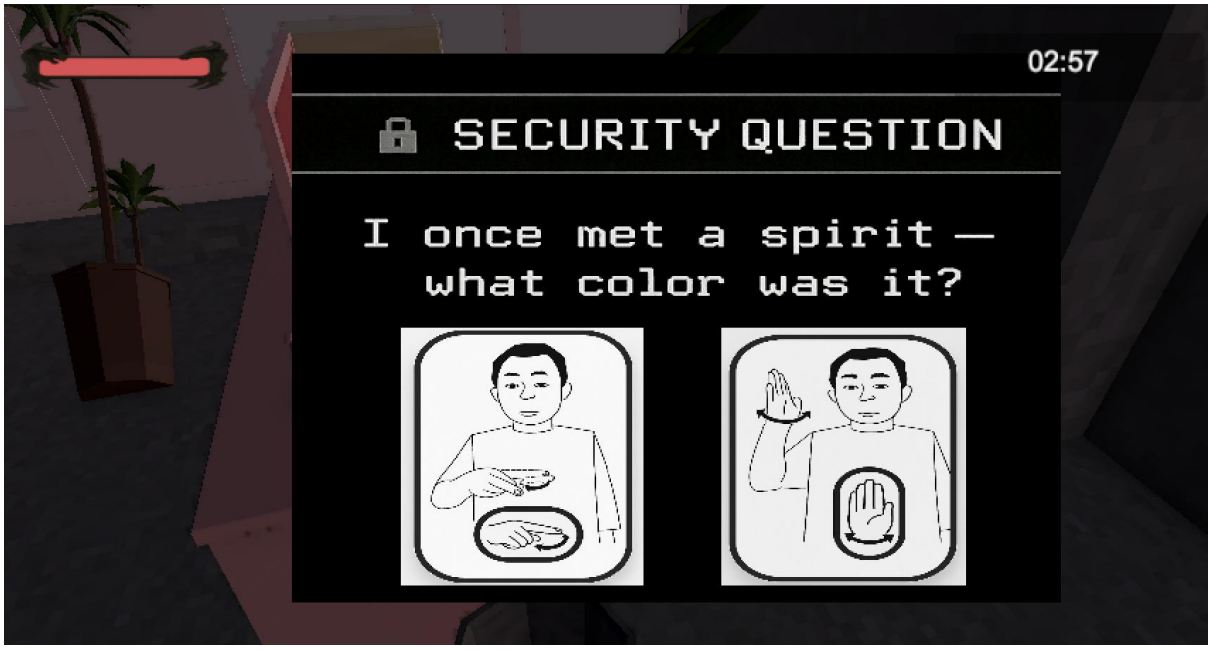


Figure 5.4.19 Security Question Interface – Sign-Language Challenge

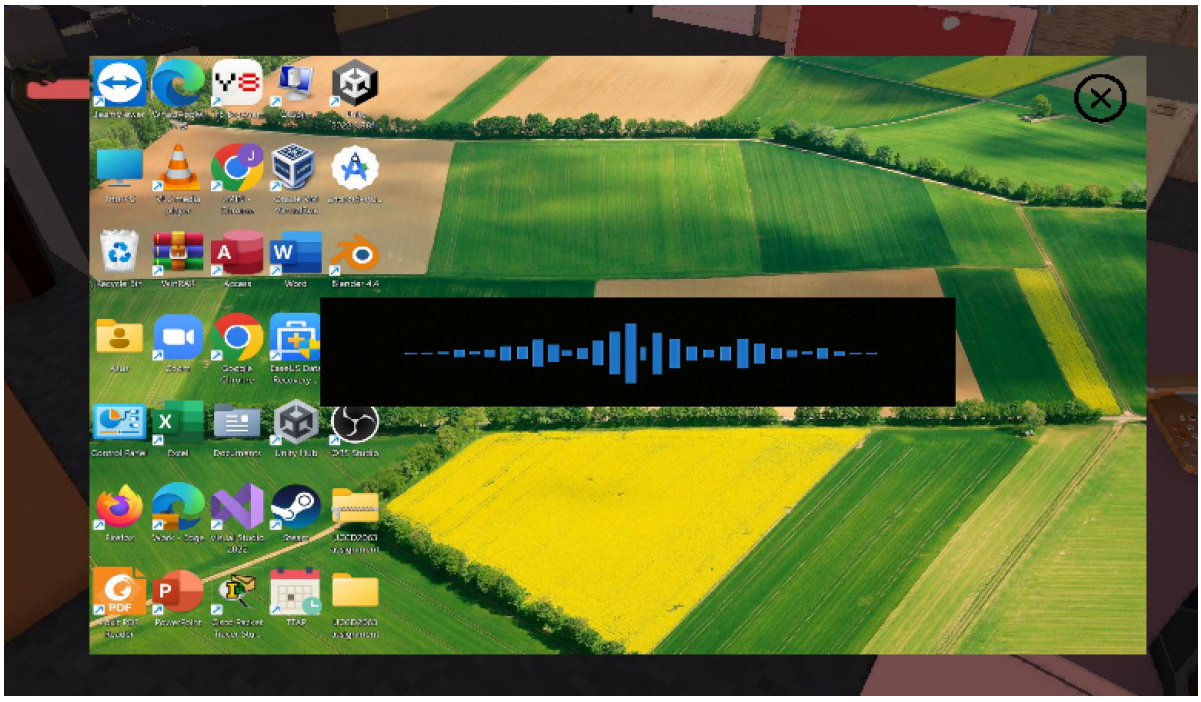


Figure 5.4.20 Voice Record Of The Way Out

Players follow the prompts to find the doctor's office. When they get close to the doctor's computer, an open Screen button will appear. Clicking it will prompt a security question. After the player successfully answers the question, the computer page will appear and the player needs to click on the audio to play it. After the audio is played, the player will know the password to exit and the password to enter.

Figure 5.4.21 Toxic Mist Trigger and Health Deduction

To simulate a hazardous laboratory environment, a **trigger zone** is placed just inside the lab entrance.

- **Trigger Logic** – When the player's collider enters the defined zone, a C# script immediately reduces the player's health value (HP) and updates the on-screen health bar.
- **Visual Effect** – A dense yellow-white **toxic-mist particle system** activates simultaneously, filling the room to convey the sensation of inhaling poisonous gas.
- **Gameplay Impact** – Continuous exposure maintains the damage over time, reinforcing the need to recall and perform the previously learned danger sign before entry.

5.4.8 Final Escape & Ending

Figure 5.4.22 Sign-Language Password Door

Figure 5.4.23 Last Scene - Back To the Scene Of Story Background

Unlike the earlier door that required a numeric code, the final barrier is unlocked through sign language. The player must recall every clue encountered along the journey and reproduce the correct sequences of gestures. Only when the full sign-language password is performed does the door slowly open. Beyond it waits the car from the opening accident, dimly lit in the night. As the player approaches, the scene fades to black, bringing the story to a quite close.

Figure 5.4.22 Victory Scene – Player Escapes in Her Blue Car

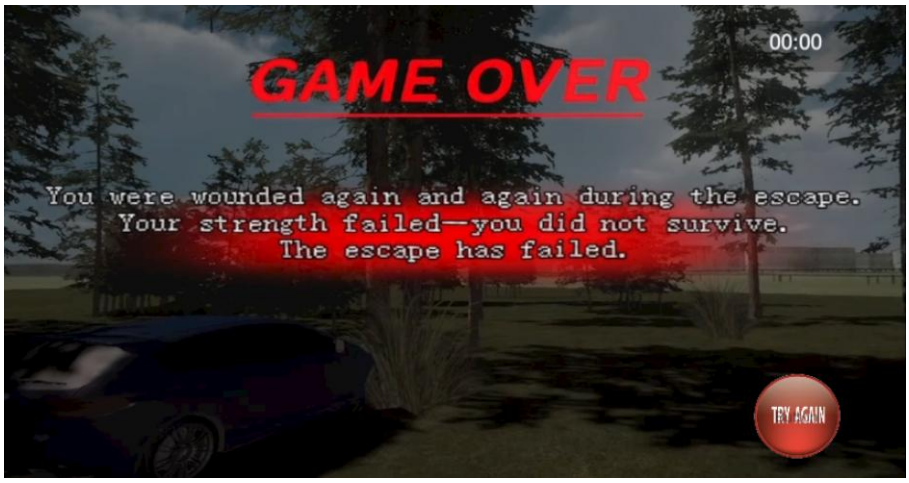


Figure 5.4.23 Game Over Scene

Endgame – Failure Path (Countdown/Health)

- **Trigger:** Countdown reaches 00:00 *or* Health ≤ 0.
- **Action:** System hides the HUD elements (Scrollbar & Countdown), plays **Game Over video** with BGM, and shows **TRY AGAIN** button (scene reload on click).
- **Result:** Player remains in the same camera context (no camera disable). *Figure 5-A. Game Over video with BGM and TRY AGAIN button.*

Endgame – Victory Path

- **Trigger:** Player opens the last door and enters the **Victory trigger**.
- **Action:** System plays **Victory video** with BGM, then shows **Win panel** (optionally proceed to next scene).
- **Narrative Link:** Visuals echo the comic-style prologue (blue car escape).

5.5 Implementation Issues and Challenges

During development, the most significant difficulty was **learning and mastering the Unity engine**. Unity was not covered in formal coursework, and its complex architecture and frequent version updates created a steep learning curve. The developer had to go far beyond basic editor operation—exploring resource management, scene construction, scripting logic, and performance optimization—often without up-to-date tutorials. Substantial time was devoted to independent research, experimentation, and troubleshooting.

A particularly demanding task was the **design of the sign-language animation and interaction system**. There were **no ready-made sign-language animation resources available**, so the developer had to **learn Blender from scratch and painstakingly create each sign-language gesture frame by frame**, an effort that was both technically challenging and extremely time-consuming.

Additional challenges included:

- Designing and validating a custom interaction mechanism so that players could follow the self-created visual prompts to solve puzzles.
- Seamlessly integrating these complex animations and interaction logic with the rest of the gameplay (monster chases, countdown events, etc.) without conflicts.

Ensuring **accuracy and naturalness** of the hand signs required repeated testing and fine-tuning of animation timing and transitions. This cross-disciplinary effort became a near-breakthrough process, demanding strong self-learning ability and continuous experimentation.

5.6 Concluding Remark

The successful completion of this project demonstrates the feasibility of combining real-time 3D game design with an educational objective—in this case, teaching basic sign-language recognition through immersive survival gameplay. Working as the sole developer, the author progressed from initial concept to a polished, fully playable build by independently mastering Unity, C# and Blender. Key achievements include

- Creation of an original sign-language interaction system with hand-crafted Blender animations

- Integration of complex triggers—monster AI, countdown events, and multi-stage puzzles—into a stable and responsive Unity environment.
- Delivery of a cinematic narrative experience with professional audio-video sequencing and smooth performance across multiple test machines.

The final product meets all objectives set out in Chapter 1: it is technically stable, visually engaging, and provides meaningful exposure to sign language within an exciting escape-puzzle framework. This project also lays a scalable foundation for future enhancements, such as expanded gesture vocabularies, proving that a single-developer effort can bridge entertainment and education effectively.

CHAPTER 6
System Evaluation And Discussion

6.1 System Testing and Performance Metrics

Test Category	Objective & Target	Measurement Method
Functional Testing	All core gameplay features—password door, sign-language hints, trigger-zone countdowns, victory and game-over videos—must operate reliably even during repeated, high-frequency interactions.	Repeated full play-throughs with rapid entering and exiting of trigger zones while monitoring script and event responses.
Frame Rate (FPS)	Maintain an average of **≥ 60 FPS** for smooth visual performance.	Unity ***Stats*** panel used to monitor real-time FPS and calculate an average.
CPU / GPU Load	CPU Main Thread average time < 16 ms; Render Thread < 10 ms.	Unity ***Stats*** panel and Profiler to record CPU Main and Render thread timings.
Scene Loading Time	From launch to first interactive frame ≤ 5 seconds.	Stopwatch measurement from application start to player control readiness.
Input Latency	Key press to on-screen action ≤ 20 ms.	High-frame-rate screen recording to measure time difference between input and visible response.

Table 6.1 System Testing and Performance Metrics Specification

6.2 Testing Setup and Result

Category	Test Details	Result / Observation
Test Devices	1. **ASUS Vivobook 15 OLED** (Intel i7-12700H, 16 GB RAM, RTX 3050 4 GB, Windows 11 Home 64-bit) 2. **Lenovo IdeaPad Slim 3i (14", Gen 9)** (Intel i5-13420H, 16 GB RAM, Intel Iris Xe + RTX 2050 4 GB, Windows 11 Home 64-bit)	Both laptops successfully ran the final build without errors or crashes.
Functional Testing	Verified all gameplay logic: password doors, sign-language input, trigger zones, countdown timer, and NPC AI. High-frequency stress: repeatedly entering/leaving trigger zones to simulate rapid interactions	All mechanics worked correctly. Doesn't have any missed or logic errors after repeated stress tests.
Performance Metrics (Unity Stats)	Average FPS and CPU/GPU timings measured in Game view. Unity **Stats Panel** reported: **82.9 FPS**, CPU main thread **12.1 ms**, render thread **6.5 ms**, ~**319k tris** & **286k verts**.	Stable frame rate well above 60 FPS on both test devices. CPU and GPU usage remained within safe limits, ensuring smooth gameplay and responsive input.
Loading & Input Latency	Timed full-scene load and in-game input response.	Scene loading completed in < 4 seconds on both devices. Input latency imperceptible (<20 ms) during all tests.

User Experience Evaluation	Two external testers completed full play-throughs. Feedback focused on control comfort, clarity of sign-language prompts, and overall immersion.	Both testers reported smooth controls, clear sign-language cues, and immersive audio-visual feedback. No motion sickness or frame stutter observed.

Table 6.1 Testing Setup and Result Specification

6.3 Project Challenges

During the testing and optimization phase, several technical issues required focused effort to achieve a stable final build:

1. Performance Bottlenecks

Early builds suffered from **frame-rate drops and long scene-loading times** due to high-resolution textures and complex scene assets.

- **Texture compression and material simplification** were applied to non-critical areas (rooftops, vehicle undersides, table bottoms) to reduce rendering load.
- **Efficient scene management and object pooling** minimized draw calls and controlled dynamically spawned objects. These optimizations significantly reduced CPU/GPU usage, delivering smooth frame rates and fast loading while preserving visual quality.

2. Trigger Reliability

Rapid, repeated player movement into and out of trigger zones initially caused **inconsistent event firing** or occasional detection failures.

- **Collider size adjustments** and **script refinements** ensured accurate detection even during high-frequency interactions.
- Functionally similar NPCs were given **independent scripts**, preventing shared-logic conflicts.

3. Sign-Language Clarity and Interaction

Maintaining the **clarity of sign-language prompts across different resolutions** required repeated UI scaling and the use of high-contrast textures.
An additional feature was added allowing players **to click on an NPC's face to auto-focus the camera and play a dedicated sign-language demonstration video**, improving both interaction quality and learning effectiveness.

6.4 Objectives Evaluation

This chapter reviews the key objectives defined in Chapter 1 and reflects on the outcomes from three perspectives: sign-language interaction, overall game performance, and feature enhancements.

6.4.1 Sign-Language Interaction

The first goal was to create a real-time sign-language interaction system that responds instantly during gameplay. Testing focused on the stability of gesture recognition, the responsiveness of trigger detection, and the clarity of on-screen prompts. Multiple rounds of testing showed that the hand-sign visuals remained sharp and easy to read, and recognition accuracy stayed high even when players quickly entered and exited trigger zones. These results confirm that the interaction system is not only reliable but also carries educational value, enabling players to naturally learn basic sign language while solving puzzles.

6.4.2 Overall Game Performance

The second objective was to integrate sign-language interaction fully into a Unity-based escape-puzzle experience. The entire flow—from collecting clues and recalling gestures to unlocking the final password door and triggering the ending sequence—underwent repeated verification. To avoid logical conflicts, individual scripts were written for each NPC and collider sizes were carefully tuned, while high-contrast UI elements ensured clear visibility across different screen resolutions. Final play-tests confirmed that every puzzle and animation executed correctly. When players clicked on an NPC, the camera automatically rotated toward the character and played the corresponding sign-language demonstration video. Test participants highlighted the intuitive controls and the cinematic finale featuring the crashed car, noting that the game successfully delivered both narrative impact and educational value.

6.4.3 Feature Enhancements and Experience Improvements

To strengthen learning outcomes and overall playability, the project introduced enhancements beyond the basic puzzle mechanics. Key improvements included:

- the ability to replay sign-language demonstrations on demand, and
- the use of simplified materials in areas invisible to the player's viewpoint—such as rooftops, the underside of vehicles, and the backs of furniture—to reduce rendering load and shorten loading times.

Testing confirmed that these optimizations run stably, keep loading times short, and do not compromise visual quality. Players reported that the replayable sign-language videos improved comprehension, while the performance tweaks maintained a smooth and visually consistent experience.

6.5 Concluding Remark

The *HUSHHH* project demonstrates that a single developer can successfully combine **sign-language education** with an **immersive horror-escape experience** on the Unity platform. Throughout development, the **developer** overcame significant challenges—self-learning advanced Unity features, creating all sign-language animations frame-by-frame in **Blender**, and optimizing performance for a large, asset-rich environment—while maintaining a clear focus on both educational value and player immersion.

Key achievements include:

- **A complete narrative flow**: from the comic-style prologue, exploration, puzzles, and monster encounters to the final victory and game-over video sequences, creating a cohesive and dramatic storyline.
- **Accurate and repeatable sign-language demonstrations**: enabling players with no prior sign-language knowledge to learn and apply essential gestures naturally during gameplay.
- **Stable system performance**: consistent frame rates and short loading times on ordinary consumer laptops, even with complex assets and expansive scenes.

Chapter 7
Conclusion and Recommendation

7.1 Conclusion

This project sets out with a clear mission: to design and develop an interactive sign-language escape puzzle game that educates while it entertains. From the earliest design sketches to the final build, every mechanic—trigger detection, sign-language clues, cinematic narrative—was crafted to immerse players in an experience where sign language is not merely a decorative element but the very key to progress.

After months of development and multiple rounds of rigorous testing, the system achieved the technical goals of stable performance, accurate gesture recognition, and intuitive interaction. Equally important, it fulfils the deeper purpose behind the project. By requiring players to observe, recall, and perform real sign gestures in order to solve puzzles, the game offers a natural introduction to sign language. Players who might never have encountered sign language are encouraged to slow down, watch closely, and repeat each movement—turning gameplay into an act of learning.

This process does more than teach a few basic signs. It fosters empathy and raises awareness of the daily challenges faced by the Deaf and hard-of-hearing community. Through its narrative of silent communication and reliance on visual cues, the game subtly invites players to experience a world where hearing is not the default. It is my hope that the game will continue to serve as a small bridge between hearing and Deaf cultures, sparking curiosity, respect, and a desire to learn more long after the credits roll.

7.2 Recommendations

Although the current version successfully delivers an engaging and educational experience, several avenues for future enhancement could broaden the game's reach and deepen its impact.

1. Expansion of Content and Levels

The present build contains a single, carefully designed escape scenario. Future iterations could introduce multiple stages with progressive difficulty, richer storylines, and a wider vocabulary of signs.

Additional characters, environmental puzzles, and branching endings would not only lengthen playtime but also expose players to a greater variety of sign-language expressions and cultural contexts.

2. Cross-Platform Deployment

Due to time and resource constraints, the project was released only as a PC-based application. Porting the game to mobile platforms such as iOS and Android would dramatically increase accessibility, allowing casual players, students, and educators to experience it without specialized hardware. A mobile version could also take advantage of device cameras and touch gestures for more intuitive sign capture.

3. Community and Educational Integration

Collaborating with Deaf organizations, sign-language instructors, and schools could provide valuable feedback for refining gestures, incorporating regional sign variations, and ensuring cultural accuracy. Adding multiplayer or classroom modes, teacher dashboards, or downloadable lesson packs could transform the game into a practical teaching aid.

4. Continuous Improvement Through Feedback

Collecting analytics and player feedback after release will highlight common learning bottlenecks and gameplay issues. Regular updates—introducing new signs, seasonal events, or community challenges—can maintain player interest and support ongoing education. By pursuing these recommendations, the game can evolve beyond its initial release into a widely accessible, continually expanding platform for sign-language awareness and appreciation—reaching a much larger audience and deepening its social and educational impact. By pursuing these recommendations, the game can **evolve beyond its initial release** into a widely accessible, continually expanding platform for sign-language awareness and appreciation—reaching a much larger audience and deepening its social and educational impact.

REFERENCES

[1]C. Veach, “Why the Disconnect between our Hearing Society and the Deaf/HoH Communities? The US Government spends Billions a year correcting an evolutionary problem that we helped to create ourselves, and this has allowed competing financial agendas to capitalize on the lack of viable communication available thr,” Linkedin.com, Jul. 22, 2020. https://www.linkedin.com/pulse/why-disconnect-between-our-hearing-society-deafhoh-clifton-veach (accessed Sep. 01, 2024)

[2]M. Abou-Abdallah and A. Lamyman, “Exploring Communication Difficulties with Deaf Patients,” Clinical Medicine, vol. 21, no. 4, Jul. 2021, doi: https://doi.org/10.7861/clinmed.2021-0111.

[3]BERNAMA, “- RAISING AWARENESS FOR THE DEAF COMMUNITY IN THE COUNTRY,” BERNAMA, Dec. 29, 2023. https://www.bernama.com/en/thoughts/news.php?id=2251189

[4]“APA PsycNet,” Apa.org, 2024. https://psycnet.apa.org/record/2017-30761-017 (accessed Sep. 01, 2024).

[5]M. Erard, “Why Sign-Language Gloves Don’t Help Deaf People,” The Atlantic, Nov. 09, 2017. https://www.theatlantic.com/technology/archive/2017/11/why-sign-language-gloves-dont-help-deaf-pe==]L. Ng, “+65 6500 6700 Fax +65 6294 8403,” 2020. Available: https://gamescom.asia/PR/PR1-gamescomasia-Launch.pdf

[7]“Insights into Malaysia’s Games Market and Its Gamers,” newzoo, Jan. 30, 2020. https://newzoo.com/resources/blog/insights-into-malaysias-games-market-and-its-gamers

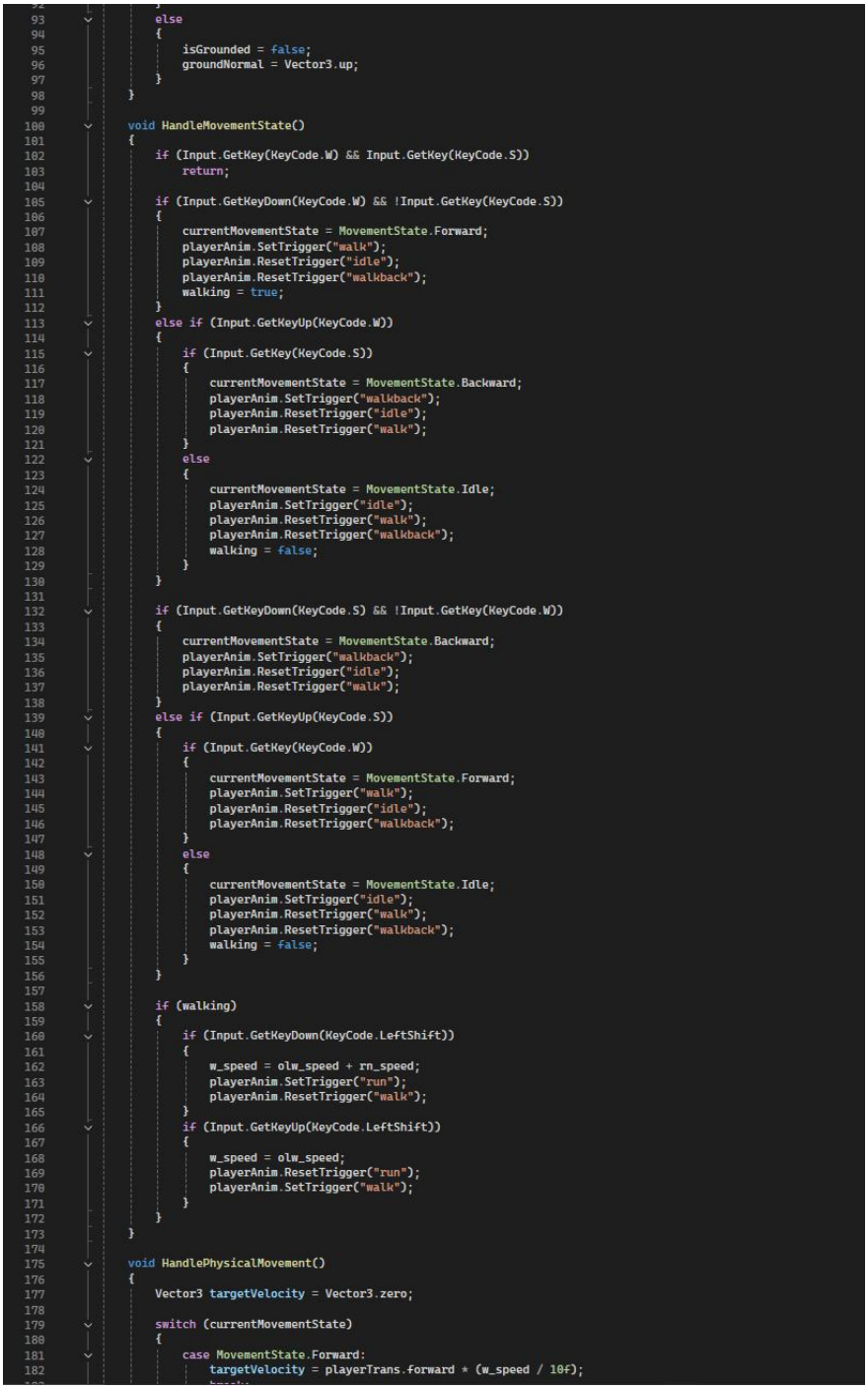
[8]“People Recall Information Better Through Virtual Reality, Says New…,” UMD. https://umdrightnow.umd.edu/people-recall-information-better-through-virtual-reality-says-new-umd-study

[9]Joy Victory , Healthy Heairing, “Hearing loss statistics at a glance,” Healthy Hearing, Dec. 07, 2017. https://www.healthyhearing.com/report/52814-Hearing-loss-statistics-at-a-glance

APPENDIX

Player Control Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class player : MonoBehaviour
{
    public Animator playerAnim;
    public Rigidbody playerRigid;
    public float w_speed, wb_speed, olw_speed, rn_speed, ro_speed;
    public bool walking;
    public Transform playerTrans;

    [Header("台阶设置")]
    public float stepHeight = 0.5f;
    public float stepDetectDistance = 0.6f;
    public float stepForce = 300f;

    [Header("斜坡设置")]
    public float maxSlopeAngle = 45f;
    public float slopeForceMultiplier = 2f;
    public LayerMask groundLayer = -1;

    // 新增：收集钻石相关
    public int collectedDiamonds { get; private set; } = 0;
    public System.Action<int> onDiamondCollected;

    private enum MovementState { Idle, Forward, Backward }
    private MovementState currentMovementState = MovementState.Idle;

    private bool isGrounded;
    private Vector3 groundNormal;
    private float targetRotation;

    void Start()
    {
        if (playerRigid != null)
            playerRigid.freezeRotation = true;
        targetRotation = playerTrans.eulerAngles.y;
    }

    void Update()
    {
        HandleRotationInput();
        HandleMovementState();
    }

    void FixedUpdate()
    {
        ApplyRotation();
        CheckGrounded();
        HandlePhysicalMovement();
        HandleStepClimbing();
    }

    void OnTriggerEnter(Collider other)
    {
        // 新增：检测钻石碰撞
        if (other.CompareTag("Diamond"))
        {
            collectedDiamonds++;
            onDiamondCollected?.Invoke(collectedDiamonds);
            Destroy(other.gameObject);
        }
    }

    void HandleRotationInput()
    {
        // 修改：只记录输入，不直接旋转
        if (Input.GetKey(KeyCode.A))
            targetRotation -= ro_speed * Time.deltaTime;
        if (Input.GetKey(KeyCode.D))
            targetRotation += ro_speed * Time.deltaTime;
    }

    void ApplyRotation()
    {
        // 新增：平滑应用旋转
        float currentY = playerTrans.eulerAngles.y;
        float newY = Mathf.LerpAngle(currentY, targetRotation, 0.3f);
        playerTrans.eulerAngles = new Vector3(0, newY, 0);
    }

    void CheckGrounded()
    {
        RaycastHit hit;
        Vector3 rayOrigin = transform.position + Vector3.up * 0.1f;

        if (Physics.Raycast(rayOrigin, Vector3.down, out hit, 1.2f, groundLayer))
        {
            isGrounded = true;
            groundNormal = hit.normal;
        }
```



```
        else
        {
            isGrounded = false;
            groundNormal = Vector3.up;
        }
    }

    void HandleMovementState()
    {
        if (Input.GetKey(KeyCode.W) && Input.GetKey(KeyCode.S))
            return;

        if (Input.GetKeyDown(KeyCode.W) && !Input.GetKey(KeyCode.S))
        {
            currentMovementState = MovementState.Forward;
            playerAnim.SetTrigger("walk");
            playerAnim.ResetTrigger("idle");
            playerAnim.ResetTrigger("walkback");
            walking = true;
        }
        else if (Input.GetKeyUp(KeyCode.W))
        {
            if (Input.GetKey(KeyCode.S))
            {
                currentMovementState = MovementState.Backward;
                playerAnim.SetTrigger("walkback");
                playerAnim.ResetTrigger("idle");
                playerAnim.ResetTrigger("walk");
            }
            else
            {
                currentMovementState = MovementState.Idle;
                playerAnim.SetTrigger("idle");
                playerAnim.ResetTrigger("walk");
                playerAnim.ResetTrigger("walkback");
                walking = false;
            }
        }

        if (Input.GetKeyDown(KeyCode.S) && !Input.GetKey(KeyCode.W))
        {
            currentMovementState = MovementState.Backward;
            playerAnim.SetTrigger("walkback");
            playerAnim.ResetTrigger("idle");
            playerAnim.ResetTrigger("walk");
        }
        else if (Input.GetKeyUp(KeyCode.S))
        {
            if (Input.GetKey(KeyCode.W))
            {
                currentMovementState = MovementState.Forward;
                playerAnim.SetTrigger("walk");
                playerAnim.ResetTrigger("idle");
                playerAnim.ResetTrigger("walkback");
            }
            else
            {
                currentMovementState = MovementState.Idle;
                playerAnim.SetTrigger("idle");
                playerAnim.ResetTrigger("walk");
                playerAnim.ResetTrigger("walkback");
                walking = false;
            }
        }

        if (walking)
        {
            if (Input.GetKeyDown(KeyCode.LeftShift))
            {
                w_speed = olw_speed + rn_speed;
                playerAnim.SetTrigger("run");
                playerAnim.ResetTrigger("walk");
            }
            if (Input.GetKeyUp(KeyCode.LeftShift))
            {
                w_speed = olw_speed;
                playerAnim.ResetTrigger("run");
                playerAnim.SetTrigger("walk");
            }
        }
    }

    void HandlePhysicalMovement()
    {
        Vector3 targetVelocity = Vector3.zero;

        switch (currentMovementState)
        {
            case MovementState.Forward:
                targetVelocity = playerTrans.forward * (w_speed / 10f);
```

```
            case MovementState.Backward:
                targetVelocity = -playerTrans.forward * (wb_speed / 10f);
                break;
            case MovementState.Idle:
                targetVelocity = Vector3.zero;
                break;
        }

        playerRigid.velocity = new Vector3(
            targetVelocity.x,
            playerRigid.velocity.y,
            targetVelocity.z
        );

        if (currentMovementState != MovementState.Idle && isGrounded)
        {
            float slopeAngle = Vector3.Angle(groundNormal, Vector3.up);
            if (slopeAngle > 3f && slopeAngle < maxSlopeAngle)
            {
                Vector3 moveDirection = (currentMovementState == MovementState.Forward) ?
                    playerTrans.forward : -playerTrans.forward;

                Vector3 slopeUp = Vector3.Cross(Vector3.Cross(groundNormal, Vector3.up), groundNormal);

                if (Vector3.Dot(moveDirection, slopeUp) > 0.1f)
                {
                    float assistForce = Mathf.Clamp(slopeAngle * slopeForceMultiplier, 0f, 50f);
                    playerRigid.AddForce(moveDirection.normalized * assistForce, ForceMode.Force);
                }
            }
        }
    }

    void HandleStepClimbing()
    {
        if (!isGrounded) return;

        Vector3 moveDirection = (currentMovementState == MovementState.Forward) ?
            playerTrans.forward : -playerTrans.forward;

        Vector3 rayOrigin = transform.position + Vector3.up * 0.1f;
        RaycastHit frontHit;

        if (Physics.Raycast(rayOrigin, moveDirection, out frontHit, stepDetectDistance, groundLayer))
        {
            float wallAngle = Vector3.Angle(frontHit.normal, Vector3.up);

            if (wallAngle > 70f && wallAngle < 110f)
            {
                Vector3 stepCheckOrigin = frontHit.point + Vector3.up * stepHeight + moveDirection * 0.1f;
                RaycastHit stepHit;

                if (Physics.Raycast(stepCheckOrigin, Vector3.down, out stepHit, stepHeight + 0.2f, groundLayer))
                {
                    float stepHeightActual = transform.position.y - stepHit.point.y;

                    if (Mathf.Abs(stepHeightActual) < stepHeight && stepHeightActual < -0.05f)
                    {
                        playerRigid.AddForce(Vector3.up * stepForce, ForceMode.Force);
                        Debug.Log($"Climbing step, height: {Mathf.Abs(stepHeightActual)}");
                    }
                }
            }
        }
    }

    void OnDrawGizmosSelected()
    {
        if (Application.isPlaying)
        {
            Gizmos.color = isGrounded ? Color.green : Color.red;
            Vector3 rayOrigin = transform.position + Vector3.up * 0.1f;
            Gizmos.DrawLine(rayOrigin, rayOrigin + Vector3.down * 1.2f);

            Vector3 moveDirection = (currentMovementState == MovementState.Forward) ?
                playerTrans.forward : -playerTrans.forward;

            Gizmos.color = Color.yellow;
            Gizmos.DrawLine(rayOrigin, rayOrigin + moveDirection * stepDetectDistance);

            Gizmos.color = Color.cyan;
            Gizmos.DrawWireCube(transform.position + Vector3.up * (stepHeight / 2),
                new Vector3(1f, stepHeight, stepDetectDistance));
        }
    }
}
```

Monster Attack Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MyEnemy : MonoBehaviour
{
    public float distance;
    public Transform Player;
    public UnityEngine.AI.NavMeshAgent navMeshAgent;
    public Animator Anim;
    private Vector3 startPosition;
    public float maxChaseDistance = 25f;



    void Start()  // 添加了缺少的 Start 声明
    {
        startPosition = transform.position;
    }

    void Update()
    {
        distance = Vector3.Distance(transform.position, Player.position);
        float fromStartToPlayer = Vector3.Distance(startPosition, Player.position);

        if (distance < 10 && fromStartToPlayer <= maxChaseDistance)
        {
            navMeshAgent.destination = Player.position;
        }
        else
        {
            navMeshAgent.destination = startPosition;
        }

        if (navMeshAgent.velocity.magnitude > 1)
        {
            Anim.SetInteger("Mode", 1); // Run
        }
        else
        {
            if (distance > 5)
            {
                Anim.SetInteger("Mode", 0); // Idle
            }
            else
            {
                Anim.SetInteger("Mode", 2); // Attack
            }
        }
    }

        public PlayerHealth playerHealth;

        public void Attack()
        {
            playerHealth.TakeDamage(10);
        }

}
```

Door Trigger And Open Script

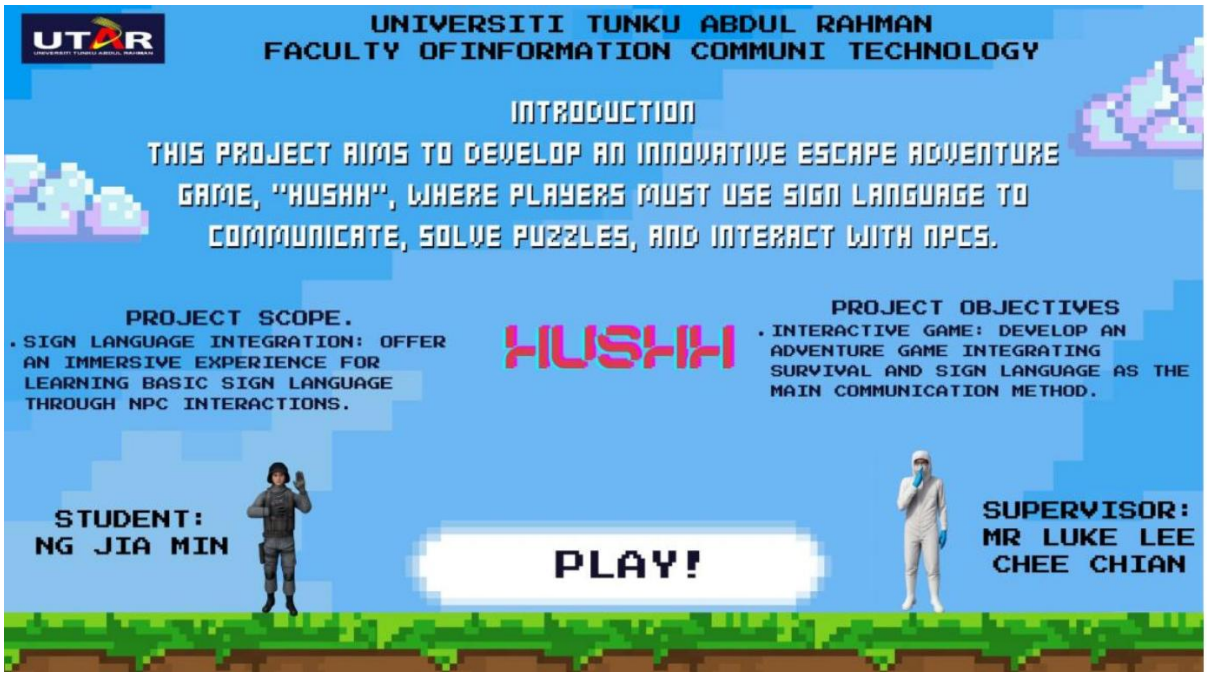
```
using UnityEngine;

[RequireComponent(typeof(Collider))]
public class PressKeyOpenDoor : MonoBehaviour
{
    [Header("References")]
    public GameObject doorObject;        // 带 Animator 的门
    public GameObject instructionUI;     // 提示面板(可空)
    public AudioSource doorOpenSound;    // 开门音(可空)

    private bool playerInRange = false;
    private bool hasOpened = false;      // ☑ 一次性触发开关
    private Animator doorAnimator;
    private Collider triggerCol;

    private void Awake()
    {
        if (doorObject) doorAnimator = doorObject.GetComponent<Animator>();
        triggerCol = GetComponent<Collider>();
    }

    private void Start()
    {
        if (triggerCol) triggerCol.isTrigger = true;
        if (instructionUI) instructionUI.SetActive(false);
    }

    private void OnTriggerEnter(Collider other)
    {
        if (hasOpened) return;                    // 已开过门就不再响应
        if (!other.CompareTag("Player")) return;

        playerInRange = true;
        if (instructionUI) instructionUI.SetActive(true);
    }

    private void OnTriggerExit(Collider other)
    {
        if (!other.CompareTag("Player")) return;

        playerInRange = false;
        if (!hasOpened && instructionUI) instructionUI.SetActive(false);
    }

    private void Update()
    {
        if (hasOpened || !playerInRange) return;  // ☑ 已开门或不在范围内都不处理

        if (Input.GetKeyDown(KeyCode.I))
        {
            OpenDoorOnce();
        }
    }

    private void OpenDoorOnce()
    {
        if (hasOpened) return;                    // 双重保险
        hasOpened = true;

        // 关提示
        if (instructionUI) instructionUI.SetActive(false);

        // 播放动画（状态名需与 Animator 中一致）
        if (doorAnimator)
        {
            doorAnimator.enabled = true;
            doorAnimator.cullingMode = AnimatorCullingMode.AlwaysAnimate;
            doorAnimator.speed = 1f;
            doorAnimator.Play("DoorOpen", 0, 0f);
        }

        if (doorOpenSound) doorOpenSound.Play();

        // 禁用触发器 & 脚本，防止再次触发或再次按键
        if (triggerCol) triggerCol.enabled = false;
        enabled = false;
    }
}
```

NPC Trigger And Animation

```
using System.Collections;
using UnityEngine;

public class dangerScript : MonoBehaviour
{
    [Header("Player Detection")]
    public Transform player;                    // 目标玩家
    public float detectionRange = 5f;           // 触发距离

    [Header("Animation Control")]
    public Animator animator;                   // 角色 Animator
    public string animationParam = "Mode";      // Animator 中的 Int 参数
    public float idleLoopDelay = 0.5f;          // Idle 循环检测间隔

    [Header("Rotation Settings")]
    public bool facePlayerDuringAction = true;  // 是否面向玩家
    private Vector3 defaultRotation;

    // 运行时状态
    private bool playerInRange = false;
    private bool actionInProgress = false;
    private Coroutine idleLoopCoroutine;

    void Start()
    {
        if (animator == null)
            animator = GetComponent<Animator>();

        defaultRotation = transform.eulerAngles;
        animator.SetInteger(animationParam, 0);  // 初始设为 Idle

        // 开始空闲循环
        idleLoopCoroutine = StartCoroutine(EnsureIdleAnimationLoops());
    }

    void Update()
    {
        if (player == null) return;

        float distanceToPlayer = Vector3.Distance(transform.position, player.position);
        bool isPlayerInRange = distanceToPlayer <= detectionRange;

        if (isPlayerInRange && !playerInRange && !actionInProgress)
        {
            playerInRange = true;
            PerformAction();
        }
        else if (!isPlayerInRange && playerInRange)
        {
            playerInRange = false;
        }
    }

    void PerformAction()
    {
        // 停止空闲循环
        if (idleLoopCoroutine != null)
        {
            StopCoroutine(idleLoopCoroutine);
            idleLoopCoroutine = null;
        }

        actionInProgress = true;

        // 直接播放 Mode = 1 动画
        animator.SetInteger(animationParam, 1);

        // 让 Animator 自己根据参数过渡
        // 如需强制立即播状态，可写实际状态名，如 animator.Play("danger", 0, 0f);

        if (facePlayerDuringAction)
            FacePlayer();

        StartCoroutine(ResetAfterAction());
    }

    IEnumerator ResetAfterAction()
    {
        string animationName = "1"; // 与参数 Mode=1 对应的状态名（如 danger）
        float timeoutDuration = 10f;
        float elapsedTime = 0f;

        do
        {
            AnimatorStateInfo stateInfo = animator.GetCurrentAnimatorStateInfo(0);
            elapsedTime += Time.deltaTime;
            yield return null;
        }
        while ((!animator.GetCurrentAnimatorStateInfo(0).IsName(animationName) ||
                animator.GetCurrentAnimatorStateInfo(0).normalizedTime < 0.95f) &&
                elapsedTime < timeoutDuration);

        // 播放结束后回到 Idle
        animator.SetInteger(animationParam, 0);
        transform.eulerAngles = defaultRotation;
        actionInProgress = false;

        // 重新启动 Idle 循环
        idleLoopCoroutine = StartCoroutine(EnsureIdleAnimationLoops());
    }

```

```
101
102     void FacePlayer()
103     {
104         Vector3 direction = player.position - transform.position;
105         direction.y = 0;
106         if (direction != Vector3.zero)
107         {
108             transform.rotation = Quaternion.LookRotation(direction);
109         }
110     }
111
112     IEnumerator EnsureIdleAnimationLoops()
113     {
114         while (animator.GetInteger(animationParam) == 0 && !actionInProgress)
115         {
116             yield return new WaitForSeconds(idleLoopDelay);
117
118             if (animator.GetInteger(animationParam) == 0 && !actionInProgress)
119             {
120                 AnimatorStateInfo state = animator.GetCurrentAnimatorStateInfo(0);
121                 if (state.normalizedTime > 0.9f)
122                 {
123                     animator.Play("0", 0, 0f); // 立即重置 Idle 动画
124                 }
125             }
126             else
127             {
128                 break;
129             }
130         }
131     }
132
133     void OnDrawGizmosSelected()
134     {
135         Gizmos.color = Color.yellow;
136         Gizmos.DrawWireSphere(transform.position, detectionRange);
137
138         if (player != null)
139         {
140             Gizmos.color = Color.green;
141             Gizmos.DrawLine(transform.position, player.position);
142         }
143     }
144 }
145
```

POSTER


UNIVERSITI TUNKU ABDUL RAHMAN
FACULTY OFINFORMATION COMMUNI TECHNOLOGY
INTRODUCTION
THIS PROJECT AIMS TO DEVELOP AN INNOVATIVE ESCAPE ADVENTURE GAME, "HUSHH", WHERE PLAYERS MUST USE SIGN LANGUAGE TO COMMUNICATE, SOLVE PUZZLES, AND INTERACT WITH NPCS.
PROJECT SCOPE.
. SIGN LANGUAGE INTEGRATION: OFFER AN IMMERSIVE EXPERIENCE FOR LEARNING BASIC SIGN LANGUAGE THROUGH NPC INTERACTIONS.
HUSHH
PROJECT OBJECTIVES
. INTERACTIVE GAME: DEVELOP AN ADVENTURE GAME INTEGRATING SURVIVAL AND SIGN LANGUAGE AS THE MAIN COMMUNICATION METHOD.
STUDENT:
NG JIA MIN
PLAY!
SUPERVISOR:
MR LUKE LEE
CHEE CHIAN