Software Engineering Department
Braude College

Capstone Project Phase B – 61999

EscapeCode - Accessible Programming Learning Game Using Unity

GitHub Link

Supervisor
Dr. Moshe Sulamy

E-mail: moshesu@braude.ac.il

Team Members

Yinon Levi

E-mail: yinon46levi@gamil.com

Shahaf Israel

E-mail: shahaf564@gmail.com

Table Of Content

1 Abstract

EscapeCode is an exciting and forward-thinking project that combines the fun and challenge of escape rooms with the opportunity to learn programming—all while being accessible to people with physical disabilities.

Using innovative technologies like eye-tracking, speech-to-text, and AI assistance, EscapeCode creates an engaging platform where players can solve coding puzzles without needing traditional input devices like keyboards or mice.

2 Introduction

In today's rapidly advancing technological landscape, the availability of diverse and innovative tools opens up countless opportunities to create groundbreaking solutions. From artificial intelligence to augmented reality, these tools not only transform industries but also reshape the way we interact with the world. However, this progress raises an important question: how can we harness these tools to revolutionize traditional fields and make them accessible to underserved populations? This challenge serves as the foundation for EscapeCode, a unique project designed to blend innovation, education, and inclusivity.

EscapeCode is an interactive desktop game that combines the excitement of escape room puzzles with the educational value of programming. Unlike traditional games or learning platforms, EscapeCode is designed to be accessible to individuals with physical disabilities. By integrating cutting-edge technologies such as eye-tracking and voice-to-text, EscapeCode allows players to engage with coding tasks without relying on a keyboard or mouse. This creates a fun, inclusive, and empowering way for users to learn programming skills, fostering both education and accessibility in one seamless experience.

The game leverages standard hardware—basic webcams and microphones—to ensure affordability and wide accessibility. Players solve puzzles by completing code blocks, guided by an AI assistant that provides hints and support throughout their journey. By offering varying levels of difficulty, EscapeCode caters to both beginners and more advanced learners, making programming education accessible to all.

2.1 Bridging the Gap: The Need for Accessible Learning Tools

For individuals with physical disabilities, learning new skills, especially programming, can be a daunting challenge. Many traditional learning platforms and tools are designed with able-bodied users in mind, creating barriers for those who cannot use a keyboard or mouse. Moreover, specialized accessibility tools, such as advanced eye-tracking devices, often come with prohibitive costs, putting them out of reach for many potential users.

This lack of inclusivity limits opportunities for individuals to gain valuable digital skills, which can improve quality of life, foster independence, and open doors to professional and personal growth. The need for accessible, affordable, and engaging educational tools has never been more pressing. EscapeCode directly addresses this gap, providing an inclusive platform that empowers users to learn programming in an interactive and enjoyable way.

2.2 Inspiration and Vision

EscapeCode draws inspiration from a variety of sources, including escape room games, educational platforms like Scratch [1] and FreeCodeCamp [2], and accessibility technologies. By combining elements from these domains, the project seeks to create a learning experience that is not only engaging but also transformative.

The vision behind EscapeCode extends beyond the game itself. It aims to demonstrate how technology can be leveraged to break down barriers, making learning accessible to everyone. The project's commitment to inclusivity, innovation, and education underscores its potential to serve as a model for future developments in accessible technology.

By providing an engaging platform for learning programming, EscapeCode aspires to inspire a broader conversation about the importance of accessibility in technology. Whether it's a student discovering coding for the first time or an individual overcoming physical challenges to gain new skills, EscapeCode stands as a testament to the power of technology to transform lives and create opportunities for all.

2.3 The Problem

Unfortunately, individuals with physical disabilities often face significant challenges when it comes to learning new skills, particularly in programming and digital literacy. For many, traditional input methods such as keyboards and mice are not viable options, which can deter them from pursuing opportunities in these fields. This exclusion not only limits their personal growth but also perpetuates a lack of diversity in the tech industry.

Beyond physical barriers, financial obstacles further exacerbate the issue. Existing accessibility tools, such as high-end eye-tracking systems or advanced speech-to-text solutions, often require specialized hardware that is prohibitively expensive for most individuals. As a result, many people with disabilities are unable to access the resources needed to develop critical skills, leaving them at a disadvantage in an increasingly digital world.

Furthermore, traditional learning platforms, while valuable, are not tailored to meet the unique needs of individuals with disabilities. For example, platforms like Scratch [1] or FreeCodeCamp [2] offer excellent programming tutorials but rely heavily on conventional input methods. This lack of adaptation makes it difficult for individuals with physical disabilities to engage with the content effectively, leading to frustration and exclusion.

The consequences of these challenges are far-reaching. Individuals who are unable to learn programming due to physical or financial constraints miss out on opportunities to improve their quality of life, gain independence, and unlock new

career paths. Moreover, society as a whole loses out on the contributions of talented individuals who could bring unique perspectives and innovations to the tech industry.

EscapeCode aims to address these challenges by providing an accessible, engaging, and affordable platform for learning programming. By utilizing readily available hardware, such as standard webcams and microphones, and integrating innovative technologies like eye-tracking and speech-to-text, EscapeCode eliminates the need for expensive specialized equipment. The game's design prioritizes inclusivity, ensuring that individuals with physical disabilities can participate fully and enjoyably in the learning process.

By breaking down these barriers, EscapeCode empowers individuals with disabilities to develop valuable digital skills, fostering a more inclusive and diverse tech community. The project represents a significant step toward a future where technology is accessible to all, regardless of physical or financial limitations.

2.4 Existing Solutions

There are currently technologies like eye-tracking devices or advanced speech-to-text solutions that enable users to type or perform actions without a keyboard. These technologies represent a significant step forward in accessibility, allowing users to interact with digital systems in ways that were previously impossible. However, most of these tools require specialized hardware, such as high-end cameras or sensitive microphones, which are costly and out of reach for many users. This financial barrier significantly limits their adoption and excludes individuals who could greatly benefit from these innovations.

Learning platforms like Scratch [1] or FreeCodeCamp [2] provide tools for learning programming, but they are not tailored for users with physical disabilities that make keyboard or mouse usage difficult. While these platforms offer engaging and effective methods for acquiring programming skills, their reliance on traditional input methods creates a significant barrier for those with physical limitations. Without alternative input mechanisms, users with disabilities may find these platforms inaccessible and frustrating to use.

Escape by CodinGame [3] is an online platform that combines coding challenges with a fun escape room experience. Players solve puzzles by writing code to progress through levels, using multiple programming languages. While the platform is both engaging and educational, it relies exclusively on conventional input methods, such as keyboards, making it less accessible for individuals with physical disabilities. This reliance highlights the broader issue of inclusivity in digital learning tools, as many existing solutions fail to consider the diverse needs of all potential users.

Although these solutions have made strides in promoting programming education, their limitations in accessibility underscore the need for more inclusive alternatives. Tools and platforms must evolve to accommodate users with varying physical abilities, ensuring that no one is left behind in the digital age. EscapeCode seeks to address this gap by providing an innovative, inclusive platform that removes barriers and empowers individuals with disabilities to learn programming in an engaging and accessible way.

2.5 Project Objectives

We plan to develop a game called EscapeCode, which integrates eye-tracking and speech-to-text technologies, designed to work with basic webcams and microphones. The game focuses on learning programming through engaging, interactive escape room challenges. Players will be able to select lines of code, move and interact with elements in the game using their eyes, and write code using voice commands. By combining these technologies, we aim to make programming education accessible to individuals with physical disabilities.

EscapeCode will go beyond merely offering accessibility; it will provide a dynamic and rewarding experience that encourages exploration and learning. The gameplay is designed to adapt to individual skill levels, ensuring that users are neither overwhelmed nor under-challenged. Players will benefit from real-time feedback and guidance from an AI assistant, which will help them progress through increasingly complex puzzles while maintaining engagement and enjoyment.

The game's design prioritizes inclusivity and affordability. By relying solely on open-source tools and readily available resources, EscapeCode ensures that the platform is accessible to a wide audience. This commitment to free and open-source technologies underscores the project's mission to eliminate financial barriers, making programming education universally attainable.

In addition to its educational value, EscapeCode aims to inspire confidence and creativity among its players. By solving puzzles and learning to code in an interactive environment, users will not only gain valuable technical skills but also a sense of achievement and empowerment. EscapeCode's ultimate objective is to demonstrate that programming is a skill that can be mastered by anyone, regardless of physical or financial limitations.

2.6 Target Audience and Stakeholders

<u>People with Physical Disabilities:</u>

EscapeCode will enable them to learn programming and develop digital skills comfortably and interactively, without the need for expensive or advanced equipment.

Teachers and Educational Institutions:

The solution can serve as a teaching tool that promotes equal opportunities and introduces innovative learning methods.

A General Audience Interested in Learning Programming:

Even individuals without disabilities can enjoy the game, which provides a fun and innovative approach to learning programming.

The Developer Community:

By using open-source and free tools, the project encourages further development of accessible solutions.

3 Literature Review

To build an accessible and interactive learning experience, EscapeCode leverages a combination of eye-tracking, speech-to-text, and AI technologies, all designed to work with basic hardware such as standard webcams and microphones. This section will present a comprehensive and detailed literature review of the technologies chosen for the project and how they align with the goal of creating an inclusive and engaging game for users with physical disabilities. Additionally, we will provide a comparison between the technologies currently available in the market and the specific tools we are using, highlighting their advantages, limitations, and suitability for our project's goals.

3.1 Eye-Tracking

Eye-tracking is a key feature of EscapeCode, allowing players to select parts of the code by looking at the screen

Eye-Tracking with Specialized Hardware

Currently, the market offers high-quality, advanced eye-tracking solutions that rely on specialized hardware. These tools are widely utilized in various fields such as research, gaming, and accessibility, where high precision and reliability are essential.

Tobii:

Tobii [4] is one of the leading companies in eye-tracking technology, offering a range of devices designed for both research and consumer applications. Their hardware provides highly accurate gaze detection and allows for sophisticated eye-tracking features, such as tracking multiple points of gaze and integrating with immersive virtual reality experiences. While Tobii devices are more expensive than webcam-based solutions, they offer exceptional precision and reliability, making them ideal for professional applications.

Pupil Labs:

Pupil Labs [5] provides open-source eye-tracking solutions that require specialized hardware, such as a pair of glasses or a camera system. These systems are highly customizable and are often used in research settings where accuracy and flexibility are crucial. The hardware can provide detailed eye movement tracking, which is beneficial for applications requiring high performance, such as detailed user behavior studies or real-time interactive feedback.

These specialized tools are ideal for scenarios where precision and advanced tracking features are essential but come with higher costs and complexity, making them less accessible for broader audiences. While they provide a high level of detail

and accuracy, they are not necessary for EscapeCode's purpose, as we prioritize accessibility and affordability.

3.1.1 Eye-Tracking Without Special Hardware

To make the game accessible to a broader audience, we sought solutions that don't require any specialized hardware and instead rely on standard webcams.

OpenCV:

OpenCV [6] is an open-source computer vision library that is frequently used for eye-tracking applications. It is a free tool that can handle eye tracking using only a basic webcam. By detecting facial landmarks, OpenCV can estimate where the player is looking on the screen. It can also be integrated with Unity [13] through additional libraries like Emgu CV [7], making it a flexible choice for real-time applications.

GazePointer:

GazePointer [8] is a simple, free, open-source solution for eye-tracking interactions, converting a basic webcam into a gaze-tracking device. It is straightforward to implement in Unity-based applications and offers basic eye-tracking features without the complexity of more advanced solutions.

Comparison:

While OpenCV [6] offers a broader range of customization and more advanced capabilities, GazePointer [8] is easier to implement and more focused on basic functionality. Depending on the specific requirements of the game, either tool can be chosen to deliver effective eye-tracking interactions. Both solutions make eye-tracking accessible without the need for specialized hardware, keeping the game affordable and user-friendly.

3.2 Turning Speech into Text

Once the player selects a line of code using their eyes, they'll complete it by speaking. This requires speech-to-text technology that's accurate, fast, and works well with standard microphones.

OpenAI Whisper:

Whisper [9] is an open-source tool that can convert speech into text with impressive accuracy. It doesn't need an internet connection, which is a big plus since it keeps everything private and works offline. Whisper is also free, making it a great fit for our project.

Vosk:

Another excellent option is Vosk [10], a lightweight, open-source speech recognition toolkit. It's quick, works offline, and integrates easily with Unity. Both Whisper [9] and Vosk are designed to handle speech from a variety of accents and environments, which means players won't have to worry about perfect pronunciation or noise.

We're leaning toward using Whisper because it's been tested widely and has excellent accuracy, but Vosk remains a strong alternative if we run into performance issues.

3.3 AI Bot Assistance

In EscapeCode, players aren't left to figure things out on their own. An AI bot will guide them through challenges, suggest solutions, and respond to commands like "Help" or "Write." This makes the game interactive and engaging.

Rasa:

Rasa [11] is an open-source tool for building conversational AI. What we like most about Rasa is that it runs entirely on the player's computer, so there's no need for an internet connection. It's also fully customizable, which means we can tailor it to understand specific commands related to gameplay.

ChatGPT API:

Another option is the free version of ChatGPT [12]. It's more advanced in terms of natural conversation and can provide dynamic suggestions. However, the free tier has limits, and we'll need to manage how much we rely on it to keep the experience smooth.

Rasa seems like the best fit because it's local, free, and reliable, but we might use ChatGPT for specific features requiring more depth.

3.4 Building the Game with Unity

Unity [13] is the backbone of our project. It's a game engine that allows us to create desktop applications with ease. Unity's free personal license includes all the features we need, from 3D modeling to advanced scripting.

Unity comes with built-in features like Visual Scripting [14], which lets us create game mechanics without writing traditional code. This is great for designing complex interactions like gaze locking and speech-based code input. The Unity Asset Store also offers free plugins for eye tracking and speech recognition, which we'll explore to speed up development.

We chose Unity because it’s user-friendly, powerful, and widely supported by free resources and tutorials.

3.5 Coding Challenges and Player Experience

At its core, EscapeCode is about solving coding puzzles in a fun and interactive way. For this, we looked at educational platforms that gamify coding to make it more engaging.

Scratch:

Scratch [1] is a free platform that teaches coding through block-based puzzles. It's simple, interactive approach is a big inspiration for how we’ll design the coding challenges in EscapeCode. Players won’t just write code; they’ll solve problems and see immediate results, much like in Scratch.

FreeCodeCamp:

FreeCodeCamp [2] is another open-source platform with structured coding lessons. Its progressive difficulty model inspires how we’ll design the game’s levels, starting with easy puzzles in the tutorial and moving toward more complex challenges as the game progresses.

Escape By CodinGame:

Escape [3] is an online platform that combines coding challenges with a fun escape room experience. Players solve puzzles by writing code to progress through levels, using multiple programming languages. While engaging and educational, it relies on traditional input methods like keyboards, making it less accessible for individuals with physical disabilities.

3.6 Why These Tools Work

Every tool we’ve chosen fits within our constraints: free to use, accessible on standard hardware, and compatible with Unity [13] for desktop applications. OpenCV [6] ensures that eye tracking works smoothly with just a basic webcam. Whisper [9] and Vosk [10] handle speech-to-text accurately without the need for internet. Rasa [11] and Unity tie everything together to create an interactive experience, while platforms like Scratch [1] inspire the game’s design.

By combining these tools, we’re confident we can create an engaging, accessible game that players will enjoy. Each piece of this puzzle supports the overall vision: a fun, coding-focused escape room game that anyone can play, no matter what equipment they have.

4 Development Stage

The development stage of EscapeCode reflects a structured, adaptive, and user-focused engineering process aimed at building an accessible programming game using unconventional input methods. This stage included the selection of appropriate tools and technologies, the definition of a flexible project methodology, and the continuous involvement of target users and stakeholders. In the following sections, we outline the methodology we adopted, how it shaped our development decisions, and how user feedback influenced the final product.

4.1 Project Methodology

The development of EscapeCode was guided by a flexible, modular, and iterative methodology designed to accommodate the unique demands of building an accessible game. While we did not adhere strictly to any one formal software development lifecycle (e.g., Scrum or Waterfall), our process most closely resembled an iterative-incremental model with strong user-centered design principles.

We divided the system into well-defined modules and tackled each one in successive development cycles. This modular approach enabled us to manage complexity by isolating distinct subsystems—such as user input, game logic, AI interaction, UI/UX design, and accessibility features—and developing them independently before integrating into a unified product.

The two-person team divided responsibilities efficiently:

- Developer A focused on the AI chat assistant, voice recognition (Whisper/Vosk), dialogue integration, and in-game interaction logic.
- Developer B handled the game interface, including menus, user settings (e.g., accessibility toggles), tutorial experience, and gaze-based input handling using OpenCV [6].

Each development cycle was approximately 2–3 weeks in duration and aimed to produce a functional unit—whether it be the first draft of the speech pipeline, a new game level, or an improved tracking mechanism. This semi-formal sprint-like structure allowed us to continuously refine the software without the rigidity of traditional sprint planning.

All development activities were tracked via GitHub, where we maintained a shared repository, managed issues, coordinated pull requests, and documented progress. Though not using a formal task board like Jira or Trello, our GitHub Issues section served as a lightweight backlog and changelog mechanism.

This adaptive methodology provided us with the agility to respond to unexpected technical challenges (e.g., limitations in webcam resolution affecting gaze precision), and allowed for gradual refinement based on user testing and feedback.

In practice, the combination of iterative development, clear module ownership, and collaborative integration proved highly effective for a small team working on a complex, real-time, accessibility-focused game.

4.2 User-Centered and Iterative Design Approach

Accessibility is not a feature—it is a core principle. In EscapeCode, we treated user-centered design not as a checkbox, but as a foundation that informed every design decision from the ground up. Our target audience includes individuals who may not be able to use traditional input devices such as a keyboard or mouse. As a result, the user interface, input mechanisms, and feedback systems were designed with empathy, simplicity, and flexibility in mind.

Before beginning full development, we conducted research into the everyday challenges faced by users with motor disabilities, and reviewed WCAG accessibility standards to inform the design of our interaction model. Instead of offering accessibility as an optional mode, EscapeCode is built around it—making features like gaze control and speech input the default interaction, not the fallback.

Our iterative design approach allowed us to evolve the game interface and control logic through cycles of prototyping, testing, and refinement. For example:

- The eye-tracking calibration interface was revised four times to improve target recognition under different lighting conditions.
- The AI bot assistant underwent multiple adjustments in how it delivers hints, adapting from a rigid text block system to a more context-aware, dynamic conversational experience.
- The tutorial flow was changed mid-development based on user confusion, replacing a text-heavy introduction with a guided, interactive onboarding level that walks the user through gaze and speech controls step by step.

Each feature was shaped not only by what we thought would work, but by what we observed actually worked in real-life usage. Even the smallest adjustments—such as increasing the hitbox size of gaze-selectable elements—were driven by the physical realities of the users we're trying to serve.

By embedding usability testing and internal feedback at each iteration, we were able to adapt quickly and minimize frustration during gameplay. This iterative, feedback-driven development cycle ensured that EscapeCode is not just functionally accessible—but truly usable and enjoyable.

4.3 Stakeholder Involvement and Feedback

Stakeholder feedback played a pivotal role throughout the development of EscapeCode. Our stakeholders fell into three primary categories:

1. Academic Stakeholders:
   Our supervisor, Dr. Moshe Sulamy, provided continuous guidance on both the engineering quality and the educational value of the product. His feedback emphasized practical deliverables, focus on real users, and compliance with project standards.
2. Target Users (End-Users):
   The most valuable insights came from potential users with real accessibility needs. We conducted informal user interviews and usability tests with individuals who had limited motor control or experience with assistive technologies. These sessions were invaluable for uncovering usability flaws that wouldn't be evident through theoretical analysis alone.
3. Development Peers and Reviewers:
   We also shared builds of the project with peers in the software engineering program, collecting usability notes, performance comments, and observations on gameplay clarity.

Examples of actionable feedback include:

- Users with limited vocal control struggled with certain keyword-triggered commands in the AI system. This led us to expand the voice command parser with synonyms and more lenient matching.
- Some users had difficulty maintaining gaze focus, prompting us to add a dwell-time configuration and an optional visual target lock indicator.
- Feedback on the tutorial experience helped us rework the onboarding process to be more interactive and less intimidating.

The feedback loop was not limited to isolated phases; rather, it was an ongoing process of observation, adjustment, and testing. We treated every stakeholder interaction as a design opportunity—and that mindset is reflected in the polish and usability of the final product.

5 Solution

5.1 System Overview

EscapeCode is a desktop-based educational game designed to teach programming through an engaging, accessible, and interactive environment—specifically tailored for users with physical disabilities. The game combines elements of escape room logic with real coding challenges, all controlled via alternative input methods such as gaze tracking and voice commands. Unlike traditional educational platforms, EscapeCode removes the need for keyboard or mouse input by integrating technologies such as UnitEye for eye tracking, Whisper [9] for speech recognition, and GPT-4o [12] for natural language interaction.

The system consists of five primary components:

- Gaze Input System: Controls on-screen interactions through eye movement, based on a modified UnitEye system.
- Voice Input System: Converts speech into text using Whisper, and forwards that to GPT-4o for contextual interpretation.
- AI Assistant: Provides responses, feedback, and hints based on the user's speech input.
- Game Engine: Unity engine responsible for rendering the world, managing logic, and handling gameplay.
- User Interface: Customized Unity UI designed for accessibility and clarity.

All components run locally, and the system does not require any cloud infrastructure. Communication with GPT-4o and Whisper is handled via controlled API calls, with audio files sent as .wav and processed synchronously.

5.2 Architecture and Technologies

EscapeCode's architecture follows a modular deployment pattern with local-only execution. Each major component operates independently but communicates through Unity's messaging system and prefab instances.

Technological Stack:

- Game Engine: Unity (C#)
- Gaze Tracking: UnitEye (modified gray-box)

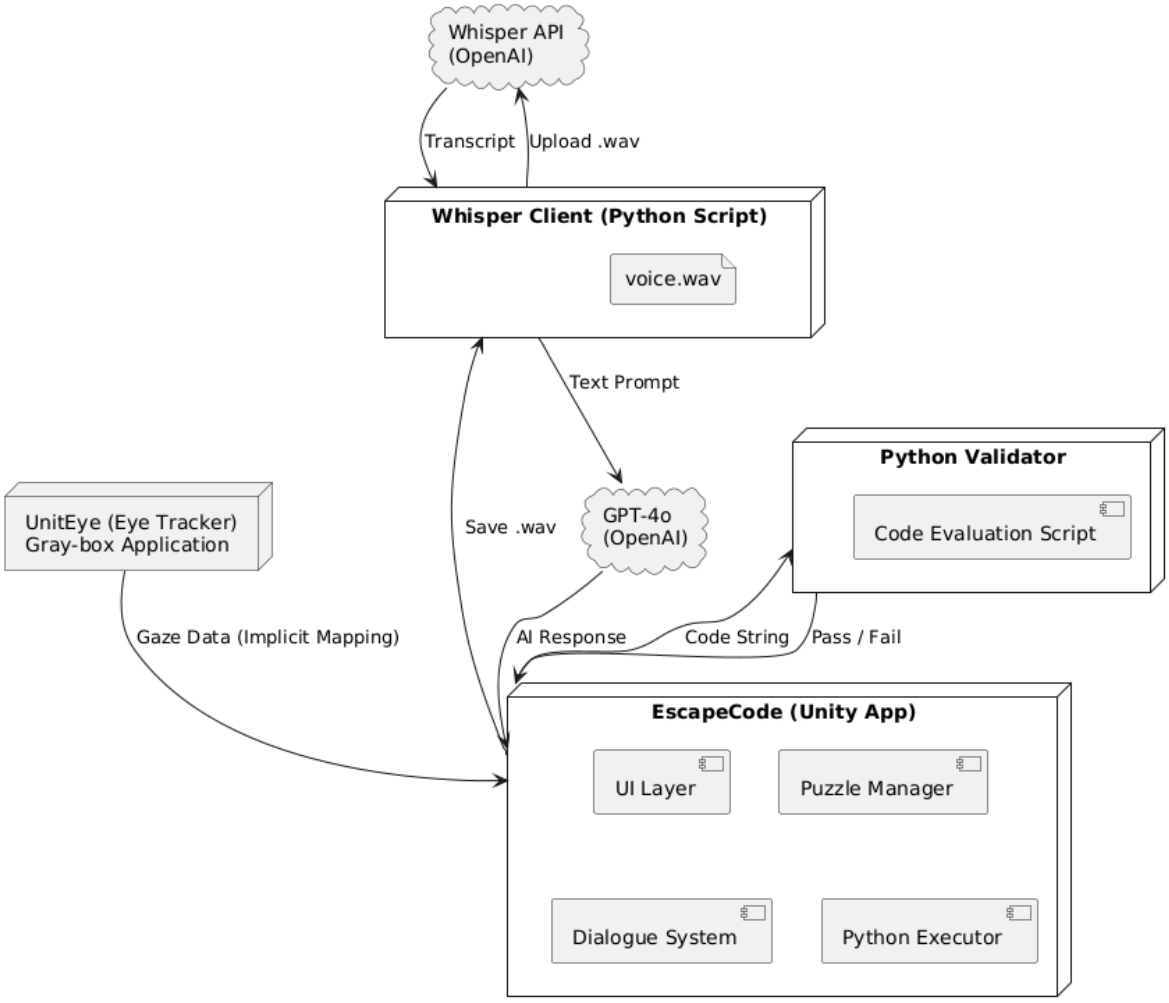
- Voice Input: Whisper (via Python script)
- Natural Language AI: GPT-4o (via OpenAI API)
- UI Layer: Unity Canvas-based interface
- Puzzle Validation: Python subprocess called by Unity to evaluate answers

Workflow Example:

1. User looks at a UI element to trigger selection (via UnitEye).
2. User enters "Code Mode" and speaks their code solution.
3. Audio is saved as .wav and processed through Whisper.
4. The recognized text is sent to GPT-4o via a predefined prompt system.
5. User submits a final code answer, which is validated via a Python script.
6. If correct, the game progresses to the next puzzle or level.

All systems run on the same machine. Communication is sequential: once one subsystem completes its task (e.g., Whisper returns text), the next begins. There is no multithreaded or real-time message queue between components.

A deployment diagram will be provided to visualize this architecture.



5.3 Activity Diagrams – System Flow

We defined several key user interaction flows in the form of activity diagrams. Two notable flows include:

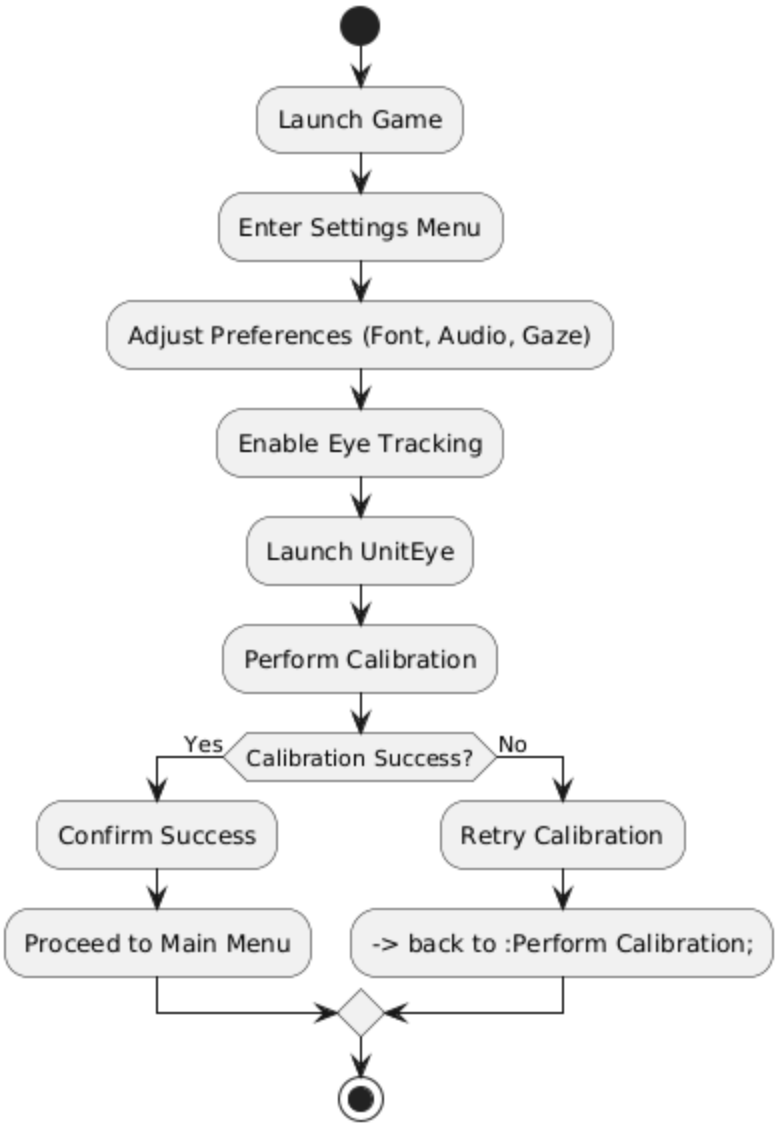
1. Game Initialization and Calibration

- Launch game
- Enter settings
- Configure preferences (font size, sound, eye sensitivity)
- Enable eye tracking
- Calibrate using UnitEye interface
- Confirm calibration success and proceed to main menu

2. Puzzle Interaction with Voice and AI

- Enter new room with coding puzzle
- Observe and understand challenge
- Enter "Code Mode"
- Speak intended solution
- Audio is recorded and passed to Whisper
- Whisper converts to text, sent to GPT-4o
- AI responds with confirmation or prompts
- Player confirms submission
- Python validator runs and checks result
- On success: proceed to next puzzle or room
- On failure: retry or request a hint

These diagrams help illustrate both user flow and system response, and are essential for demonstrating the full accessibility pipeline.



5.4 Game Logic and Algorithms

EscapeCode's game logic revolves around a challenge-response system. Each puzzle defines a set of valid outputs (via Python), and users must submit spoken solutions that are semantically correct, not just syntactically matching a single string.

The AI assistant operates on a rule-based prompt system in GPT-4o [12], where speech input (after Whisper transcription) is processed in context. Depending on whether the player is in normal conversation or "Code Mode," the assistant either guides the player or treats their message as a solution attempt.

Hints are predefined per puzzle and provided on request. There is no dynamic generation of hints by the AI.

Validation is handled by a Python script that executes the submitted code and compares its output to expected results, allowing flexibility in syntax and logic.

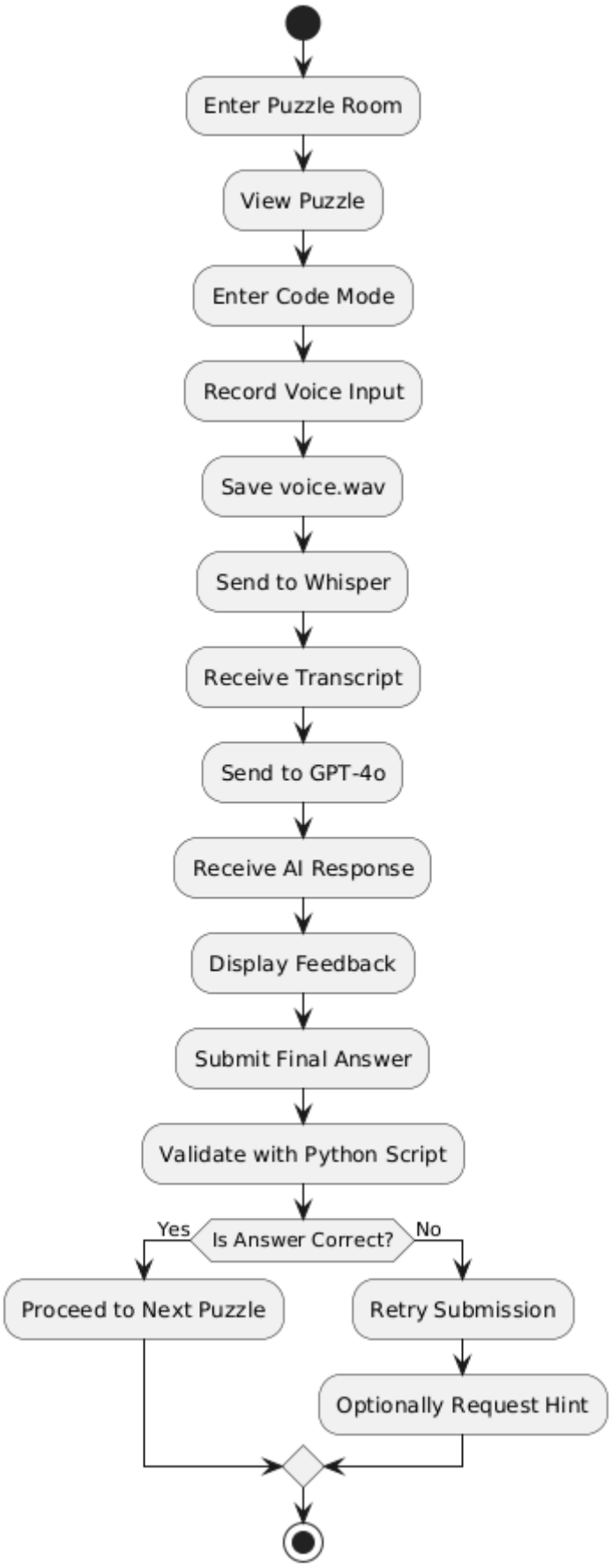
5.5 Modular Implementation Structure

The Unity [13] project is divided into discrete, focused scripts aligned with specific responsibilities:

- PuzzleManager: Controls puzzle lifecycle, solution state, and progression.
- GazeController: Interfaces with UnitEye and maps gaze to UI selections.
- DialogueSystem: Manages all interactions with the AI Assistant and coordinates speech-to-text logic.
- UIManager: Updates visual elements, menus, and accessibility preferences.
- AudioRecorder: Captures microphone input and formats it as .wav files.

The project uses Unity prefabs extensively for modularity—especially for UI panels, interactable objects, and reusable puzzle templates. While separation of concerns is maintained in many areas, some scripts combine UI and logic where practical (e.g., puzzle-specific interactions).

No advanced Unity tools like ScriptableObjects were used heavily; the focus was on clarity and maintainability.


Enter Puzzle Room
View Puzzle
Enter Code Mode
Record Voice Input
Save voice.wav
Send to Whisper
Receive Transcript
Send to GPT-4o
Receive AI Response
Display Feedback
Submit Final Answer
Validate with Python Script
Is Answer Correct?
Yes
No
Proceed to Next Puzzle
Retry Submission
Optionally Request Hint

6 Testing Stage

6.1 Evaluation Strategy

The evaluation of EscapeCode focused on ensuring both functionality and usability for users with and without physical limitations. The game was tested iteratively throughout development, using a blend of unit testing, integration testing, and real-user observation. Each new feature or subsystem was validated independently before being integrated into the main build.

A major goal of our evaluation process was to confirm the accessibility of gaze and voice input mechanisms across a range of real-world conditions, including background noise, lighting variability, and user movement.

6.2 Verification and Usability Tests

Verification focused on confirming that each component of the system met its intended functionality:

- Gaze selections activated the correct in-game buttons
- Whisper consistently transcribed clear speech to text
- The AI assistant interpreted prompts correctly depending on the game mode
- Python scripts validated code submissions accurately

Usability testing was conducted with participants, including those unfamiliar with programming and those using accessibility settings. Feedback led to several improvements such as increasing dwell time targets, simplifying UI layout, and improving in-game feedback messages.

6.3 Full Test Scenarios and Test Cases

Representative test cases included:

- Scenario 1: Player starts the game, calibrates eye tracker, and enters the tutorial.
- Scenario 2: Player enters a puzzle room, submits a solution, and receives feedback.

- Scenario 3: Player makes a voice input error and is able to retry or view the transcribed text to self-correct.
- Scenario 4: Player requests a hint and receives a predefined contextual clue.
- Scenario 5: Player completes all puzzles in a room and transitions to the next level.

Each scenario was tested with a checklist of expected states and fallback behavior.

6.4 Testing Frameworks and Tools

Although no dedicated automation frameworks were used, the following tools supported testing:

- GitHub: Issue tracking and test planning
- Unity Editor: Debugging and play-mode test iteration
- Manual test scripts: Used to ensure consistent walkthroughs of key flows
- Voice/audio review: Manual comparison of speech input and Whisper output
- Python scripts: Evaluated code correctness and reported structured errors

6.5 Accessibility and Compliance Testing

EscapeCode was evaluated against basic WCAG-inspired accessibility principles:

- Large clickable areas for gaze targeting
- High-contrast color palette
- Optional font-size and sensitivity adjustments
- Visual feedback for all inputs (audio transcription shown before confirmation)

Manual compliance testing was done with real users and simulated edge cases (e.g., low mic quality, user fatigue, misalignment during eye tracking). Results indicated that the system maintains core functionality and clarity even under non-ideal conditions.

7 Challenges

Throughout the development of EscapeCode, we encountered a range of technical, design, and integration challenges. Addressing these issues required both creative problem-solving and pragmatic compromises due to constraints such as hardware limitations, development tools, and time.

7.1 Understanding and Adapting UnitEye

One of the central challenges was working with the UnitEye eye-tracking system. As a black-box application, it offered limited documentation and minimal transparency. To tailor its performance to our game, we had to experiment with internal configuration files, reverse-engineer behavior, and conduct repeated tests. The learning curve was steep, but we successfully adapted UnitEye into a gray-box solution that functioned reliably within our specific game flow.

7.2 Audio Click System and Whisper Integration

We aimed to implement an Audio Click System that would allow players to trigger interactions with short vocal commands or clicks. Initially, we experimented with a local lightweight model (Tiny Whisper [16]), but its recognition accuracy was poor and inconsistent. Ultimately, we shifted to the full Whisper pipeline via OpenAI, but this created new challenges around latency, file handling, and smooth Unity [13] integration. Handling .wav creation, external script execution, and voice buffer management required careful timing and retries to achieve an acceptable user experience.

7.3 Git-to-Unity Workflow

Coordinating the development process across different environments posed another challenge. Unity [13] projects are sensitive to file structure, versioning, and serialization. Syncing updates between Git and Unity often caused prefab or scene inconsistencies, requiring manual conflict resolution. We developed conventions to reduce merge issues and ensured frequent communication between developers to maintain consistency.

7.4 Adapting for User Feedback

Real users played an active role in shaping the experience. Several features—including UI layout, gaze dwell timing, and interaction logic—were changed based on feedback from participants with motor impairments. Although most suggestions were helpful, they often required redesigning logic or modifying prefabs, leading to time-consuming reimplementation. Balancing user-centric adjustments with technical constraints remained a recurring challenge.

8 Lessons Learned

8.1 What Worked

One of the most significant successes in the project was the successful integration of multiple advanced technologies—such as eye tracking, voice recognition, and AI conversation systems—without any dedicated budget. These are typically expensive tools and difficult to combine, especially for teams without prior experience. The fact that we managed to implement and align them within a cohesive gameplay experience is a major achievement.

Another key success factor was the strong collaboration between team members. Each member contributed constructively, with patience, shared responsibility, and a unified goal of building a meaningful and polished product. Despite the complexity of the technologies used, the team demonstrated a high level of coordination and commitment throughout the project.

Every tool that was successfully integrated, from UnitEye to Whisper to GPT-4o [12], exceeded expectations—especially considering that this was the team's first venture into game development and accessibility-based interaction design.

8.2 What We Would Do Differently

From an organizational perspective, we recognize that the integration of technologies could have been better planned from the beginning. For example, integrating Whisper with the Audio Click System posed unforeseen challenges. In hindsight, a more deliberate architectural design and early-stage experimentation would have helped reduce time spent on debugging and adapting workflows.

While we are generally satisfied with the selected tools, limited experience at the start of the project made it difficult to assess long-term suitability. Given the chance to start again, we might still choose the same stack—but with a more focused plan for managing their limitations.

We also invested considerable time in trying to implement the Audio Click System with a local Tiny Whisper [16] model, which ultimately proved too inaccurate. This time could have been saved by going directly to the more robust cloud-based Whisper API.

8.3 Insights for Future Projects

This project offered valuable lessons in complex system integration and Unity-based game development. We gained hands-on experience working with real-time input pipelines, asynchronous processes, and prompt engineering with AI models.

Working with black-box tools like UnitEye taught us the importance of flexibility and reverse-engineering skills. Sometimes, achieving a working solution means "opening the box" and modifying configurations beyond what's officially supported.

For future accessibility-focused projects, our strongest recommendation is to prioritize feedback from target users as early as possible. Listening to their needs—and being willing to change your design accordingly—can make the difference between a theoretical solution and a truly inclusive experience.

9 Project Goal Completion

9.1 Objectives vs. Results

The original goals of EscapeCode included building a fully functional educational game that is playable using only eye-tracking and voice input. Specific deliverables were: seamless integration of gaze tracking via standard webcams, voice-to-text programming using offline models, an AI assistant for guidance, and a full set of escape-room-style coding challenges with increasing difficulty levels.

While some of the implementation methods evolved during development, the core objectives were met. For example, instead of using OpenCV [6] or GazePointer [8], we successfully adapted UnitEye into a gray-box solution. Similarly, we used the cloud-based Whisper API [9] rather than a local model to achieve better voice recognition. Ultimately, we delivered a fully playable desktop game that supports users with physical disabilities and integrates all the promised accessibility features.

9.2 Discussion of Success Metrics

Success was defined by several criteria: the game should run on standard hardware, be usable by non-programmers, support users with physical disabilities, and maintain high interaction accuracy.

Based on internal testing and user feedback:

- All core systems functioned reliably (gameplay, gaze tracking, voice input, AI interaction).
- Players were able to navigate and complete tasks without needing a keyboard or mouse.
- Feedback from a small user group confirmed the system was intuitive and engaging.

Although a full-scale usability study was not performed, feedback from early testers supported the conclusion that the project met its functional, usability, and accessibility goals.

9.3 Gaps and Unfinished Features

While the core gameplay loop was completed, a few planned features were left out due to time and resource constraints:

- A multi-tiered level system with advanced stages was not fully implemented.
- Gaze tracking could benefit from further tuning to improve precision.
- Expanding the game with more puzzles and varied content was postponed.

Despite these omissions, the final product demonstrated the full concept and validated the technical approach. These features represent natural opportunities for future work or external contributions.

10 User Guide

10.1 System Requirements

Minimum Requirements**:**

- Operating System: Windows 10 (64-bit)
- Processor: Intel Core i5 (4th Gen) or equivalent
- Memory: 8 GB RAM
- Graphics: Integrated GPU (Intel HD 5000 series or higher)
- Storage: 2 GB of free space
- Input Devices:
    - Webcam (720p minimum, for eye tracking)
    - Microphone (basic USB or built-in)

Recommended Requirements:

- Operating System: Windows 10 or 11 (64-bit)
- Processor: Intel Core i7 (8th Gen) or AMD Ryzen 5
- Memory: 16 GB RAM
- Graphics: Dedicated GPU (NVIDIA GTX 1050 or better)
- Storage: SSD with 2 GB free
- Internet Connection: Required for Whisper/GPT features (API calls)
- Input Devices:
    - Webcam (1080p recommended for better tracking accuracy)
    - Microphone (noise-canceling preferred)

10.2 Installation Instructions

Step 1: Download the Game

- Download the EscapeCode installer or ZIP package from the official distribution source (e.g., GitHub or external drive).
- Extract the contents to a folder of your choice (if ZIP).

Step 2: Check Requirements

- Ensure your system meets the minimum requirements (see “System Requirements” section).
- Verify that a webcam and microphone are connected and working.

Step 3: Run the Game

- Navigate to the installation folder.
- Double-click EscapeCode.exe to launch the game.

Step 4: Enable Accessibility Features (Optional but Recommended)

- On first launch, go to Settings.
- Enable Eye Tracking if you wish to control the game using your eyes (requires a webcam).
- Enable Voice Input to use speech for puzzle solving (requires microphone and internet access).

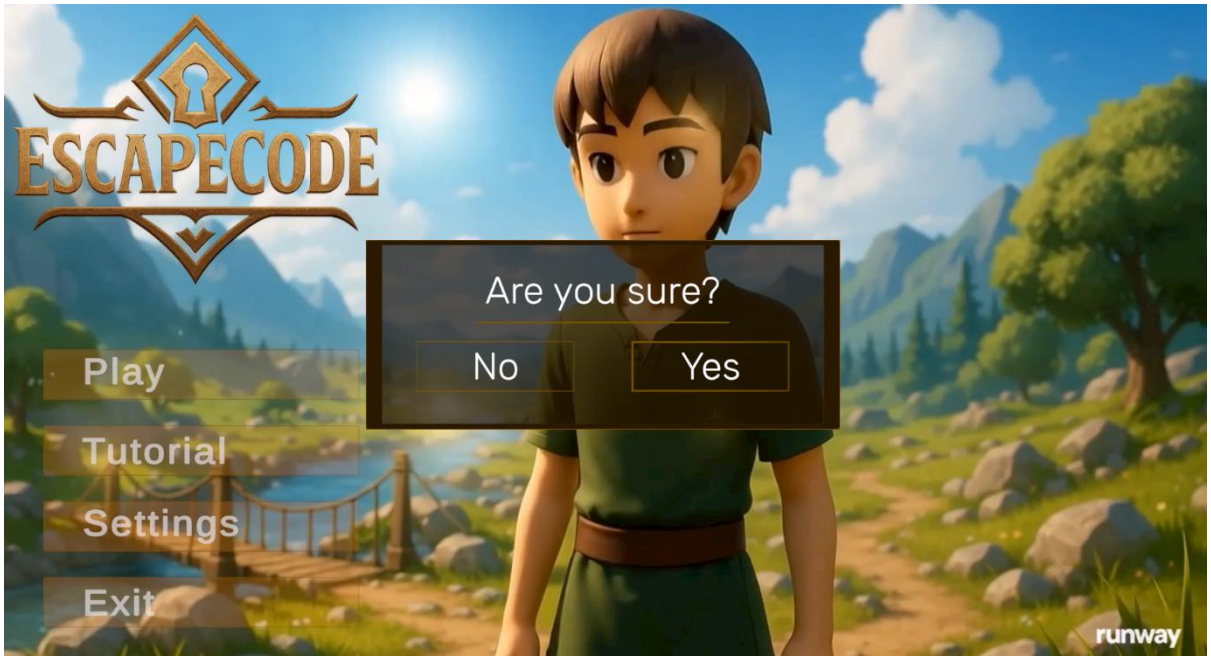
Step 5: Begin Playing

- Follow the on-screen tutorial to learn how to play using your chosen input method.
- Use “Code Mode” to start solving challenges and enjoy the experience!

10.3 Operational Workflow (Typical Use Case)

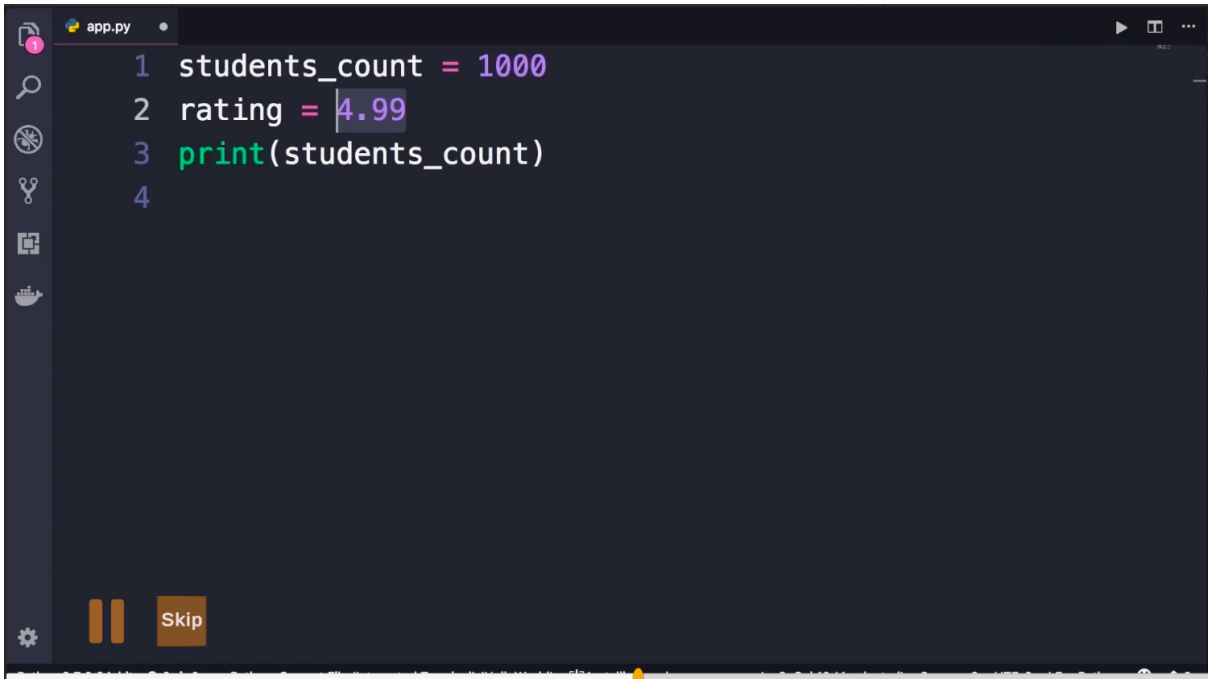
Open Game:

Clicking exit button, click Yes to exit game completely or No to stay:

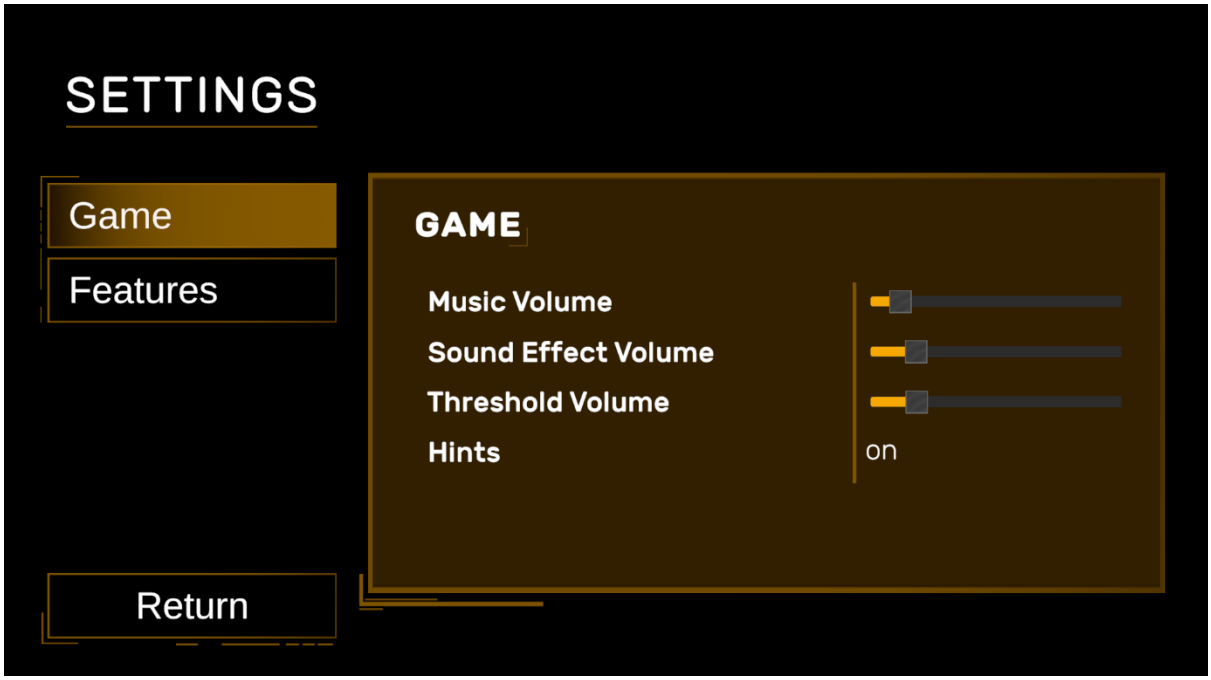


Click tutorial button in main menu to watch python tutorials.

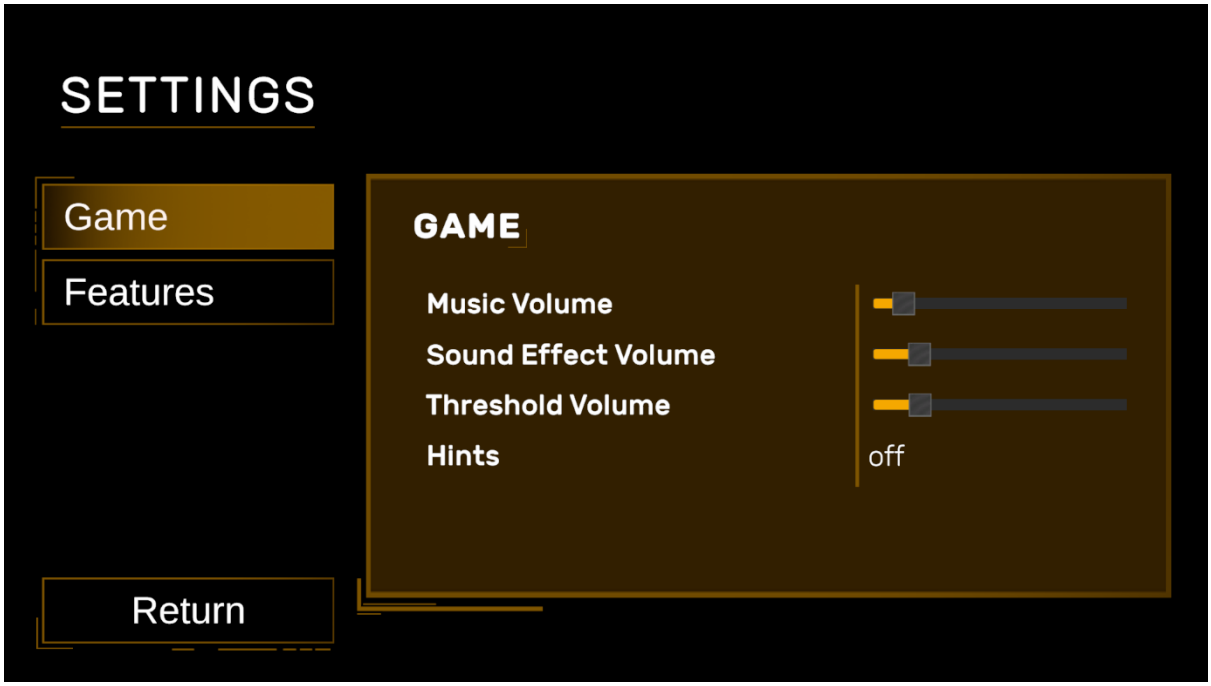
Click the pause\play button to stop or start the video accordingly. Click on the skip button to skip the video. You also can move the slide bar to navigate to a specific point in the video.



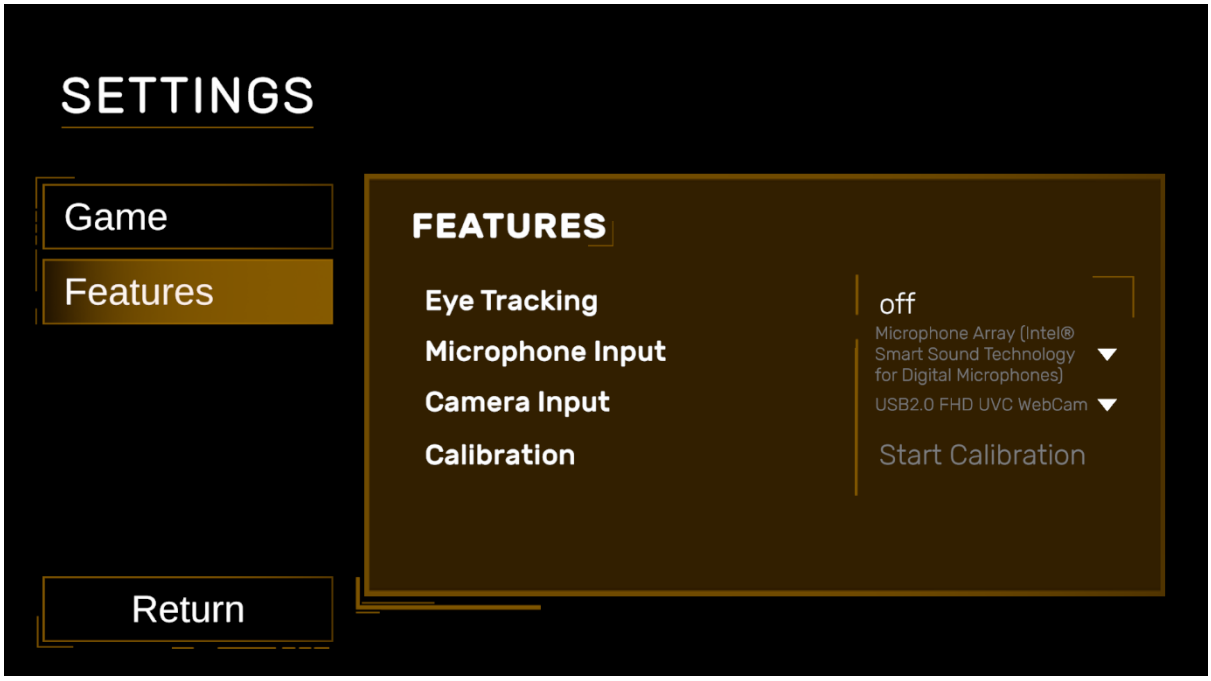
Click settings button in main menu to adjust volume preferences and decide if hints are wanted in the game:



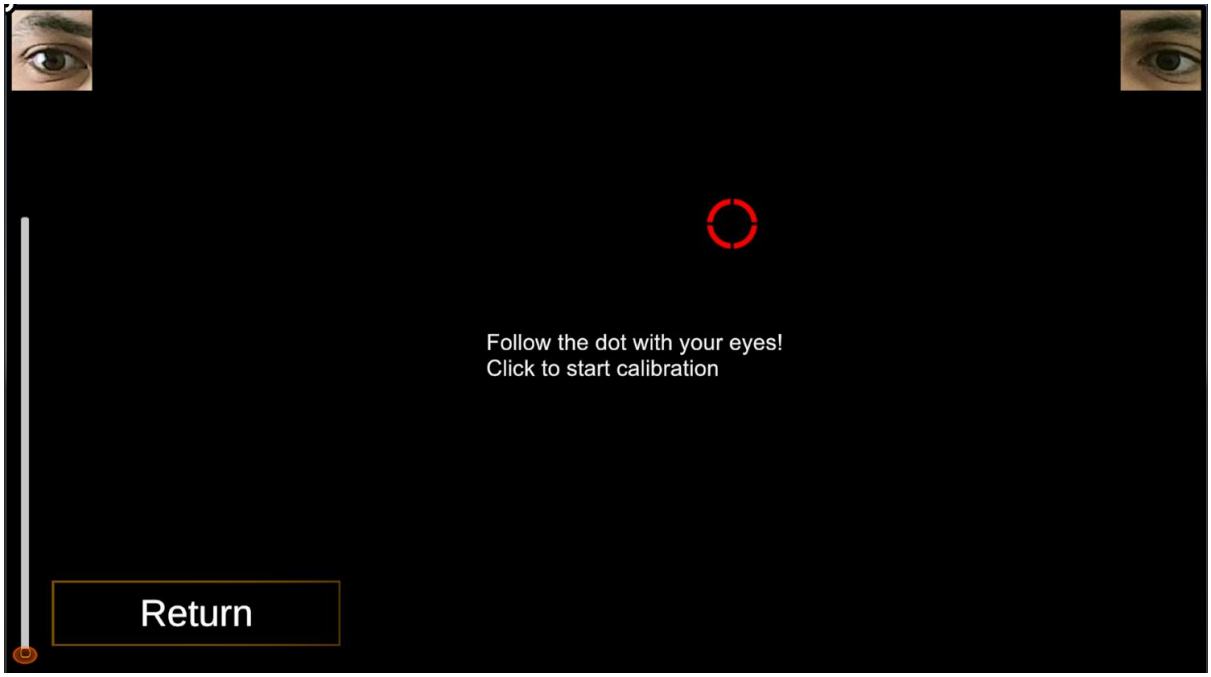
Clicking on hints button to turn off:



Clicking features tab to turn on or off eye tracking feature, when enabling it, you can choose preferred microphone, camera from dropdowns of available devices:

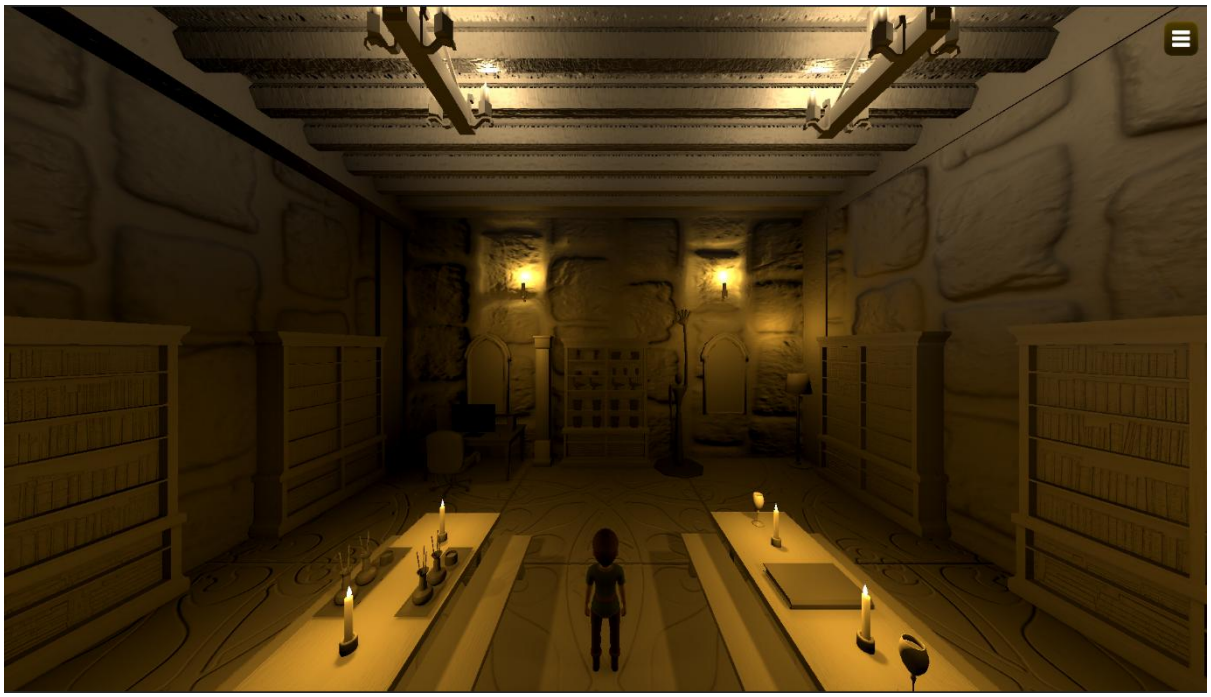


When you turn on the eye tracking feature the calibration screen shows up and you need to complete the process. Normally the calibration process will take 2-3 minutes to complete, after this you can play using your eyes to navigate the screen like a regular mouse and use any sound to make a click button. Click return to abort the eye tracking feature.

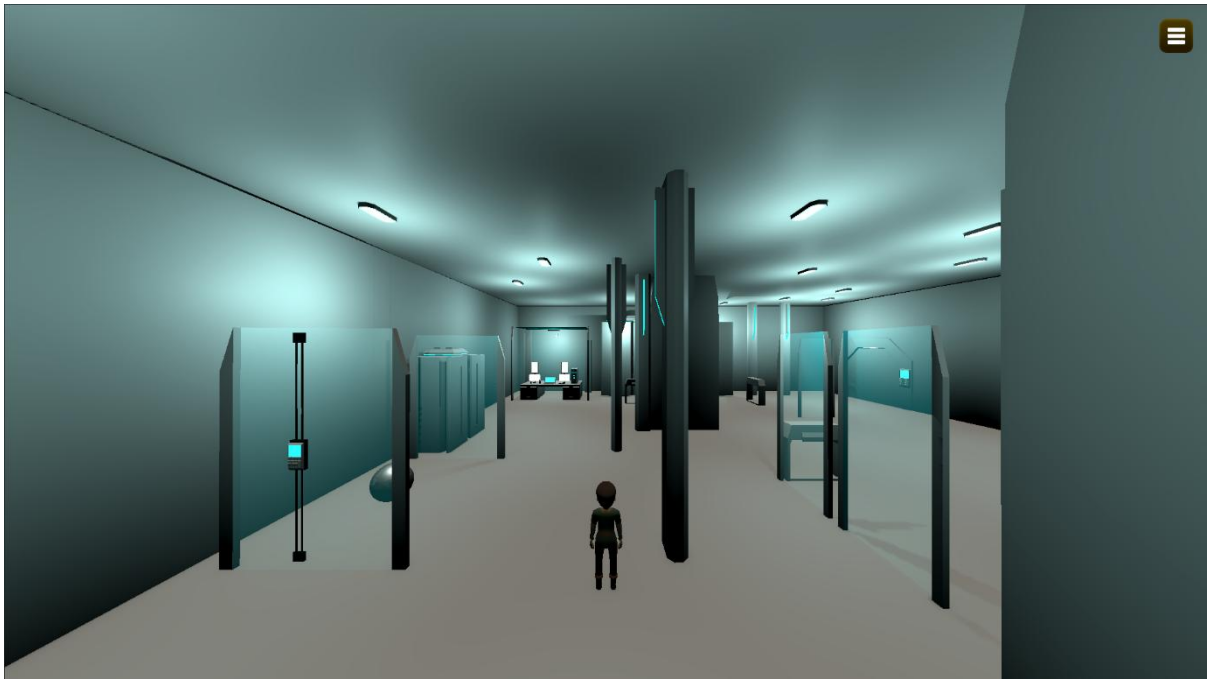


Click play and choose your desired room level:

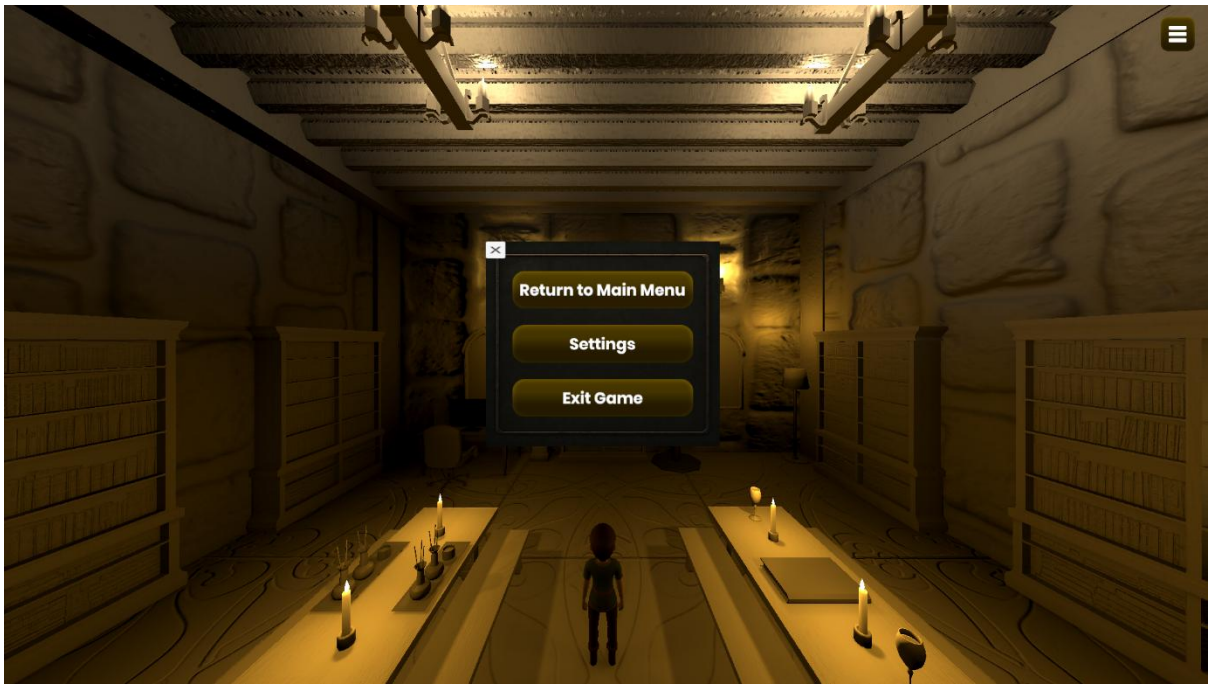
Click level 1 opens the first room (easy level)



Click level 2 opens the second room (medium level)

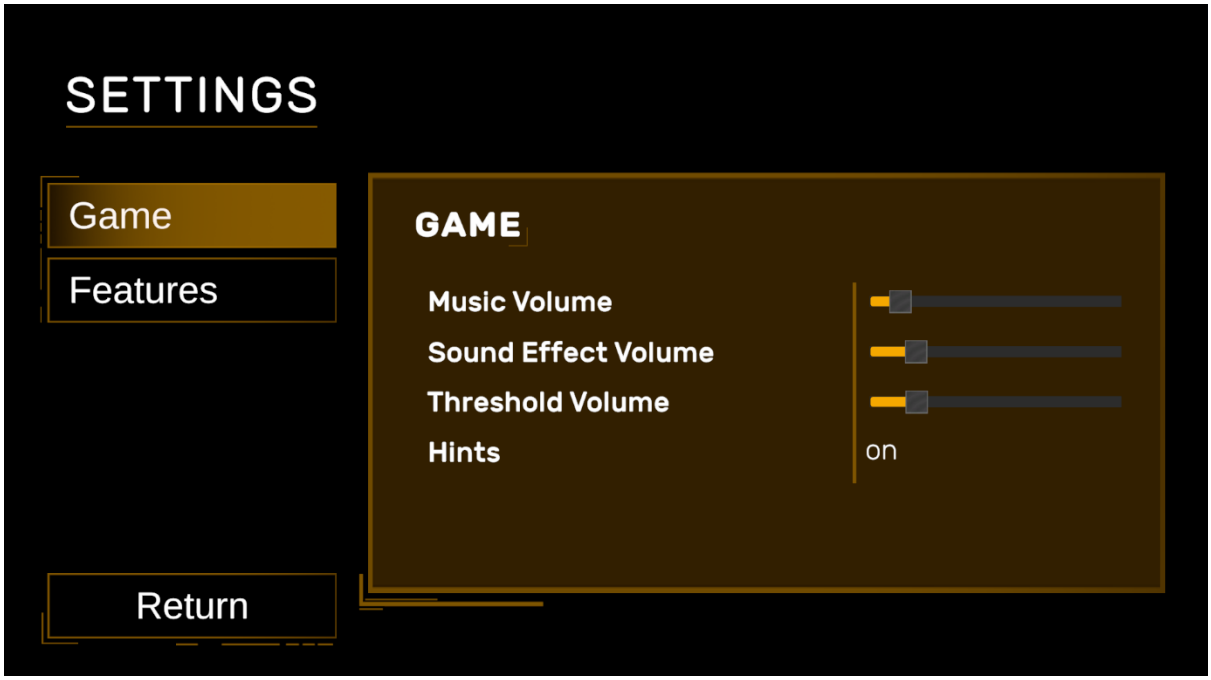


Click hamburger icon opens small menu where you can go back to main menu, go to adjust settings or exit game:



Clicking Return to Main Menu:

Clicking Settings:

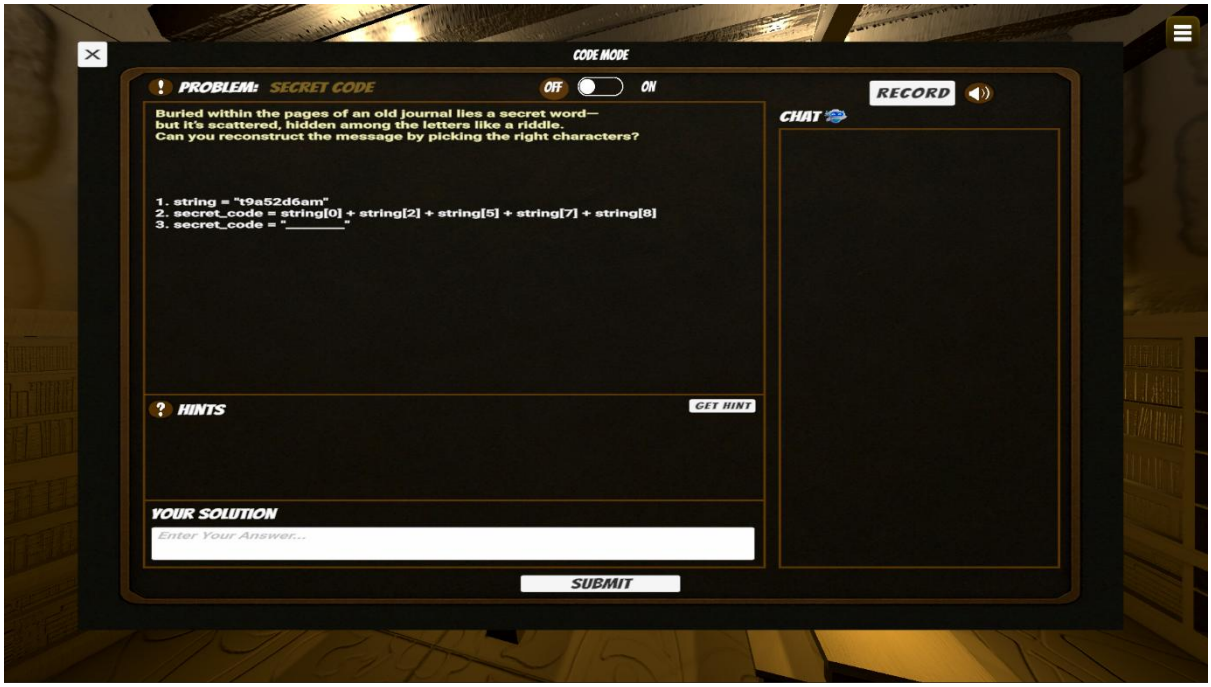


Clicking Exit Game:

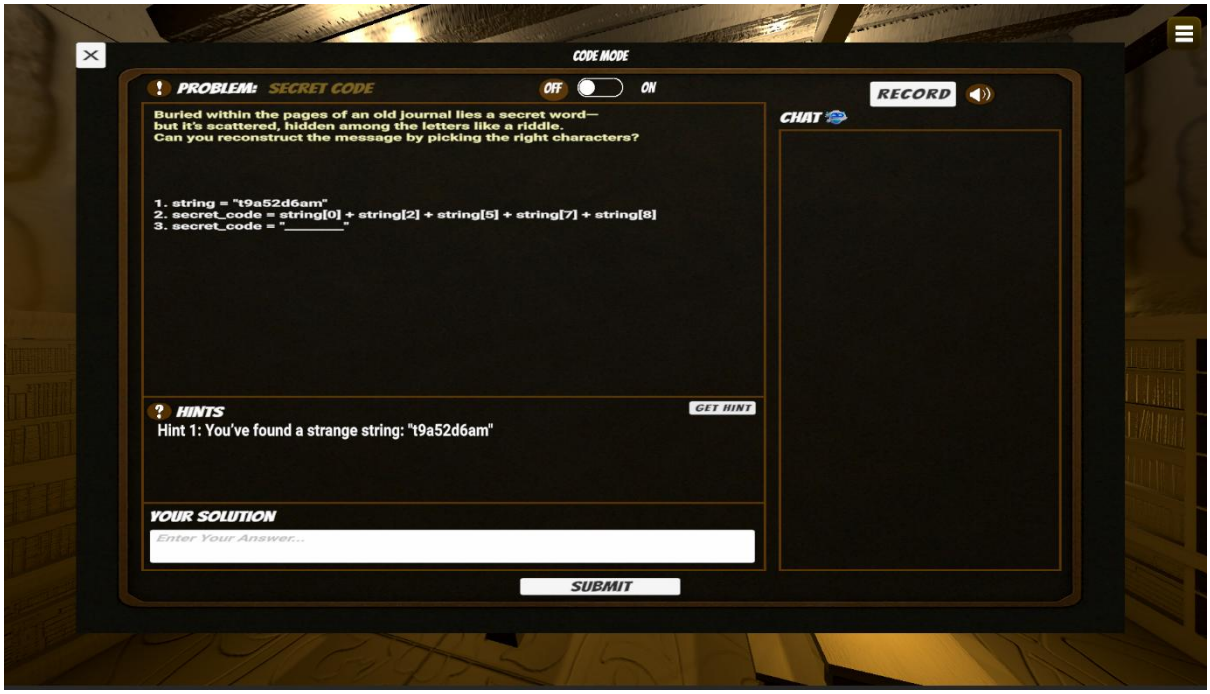
Click anywhere to move:

The Hint Glow System activates after 1 minute and 30 seconds of inactivity by displaying a small yellow circle above the object the player needs to interact with.

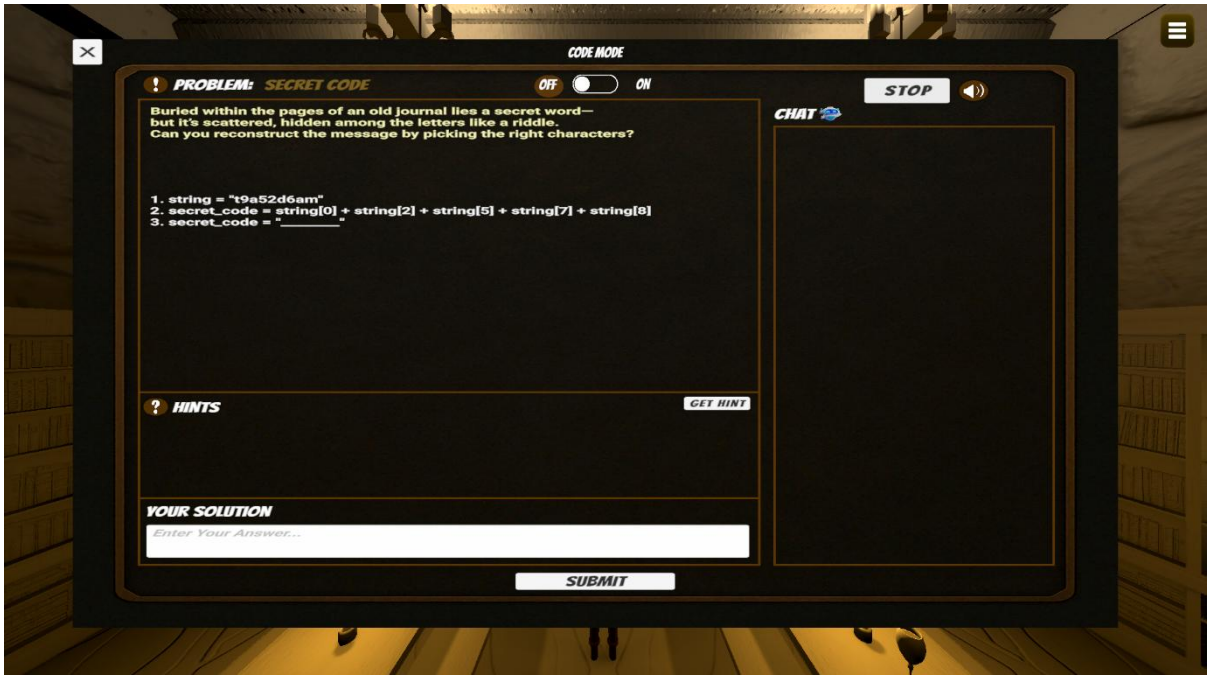
Click on object puzzle opens a window where the player needs to solve the code problem by typing/saying the right line code:



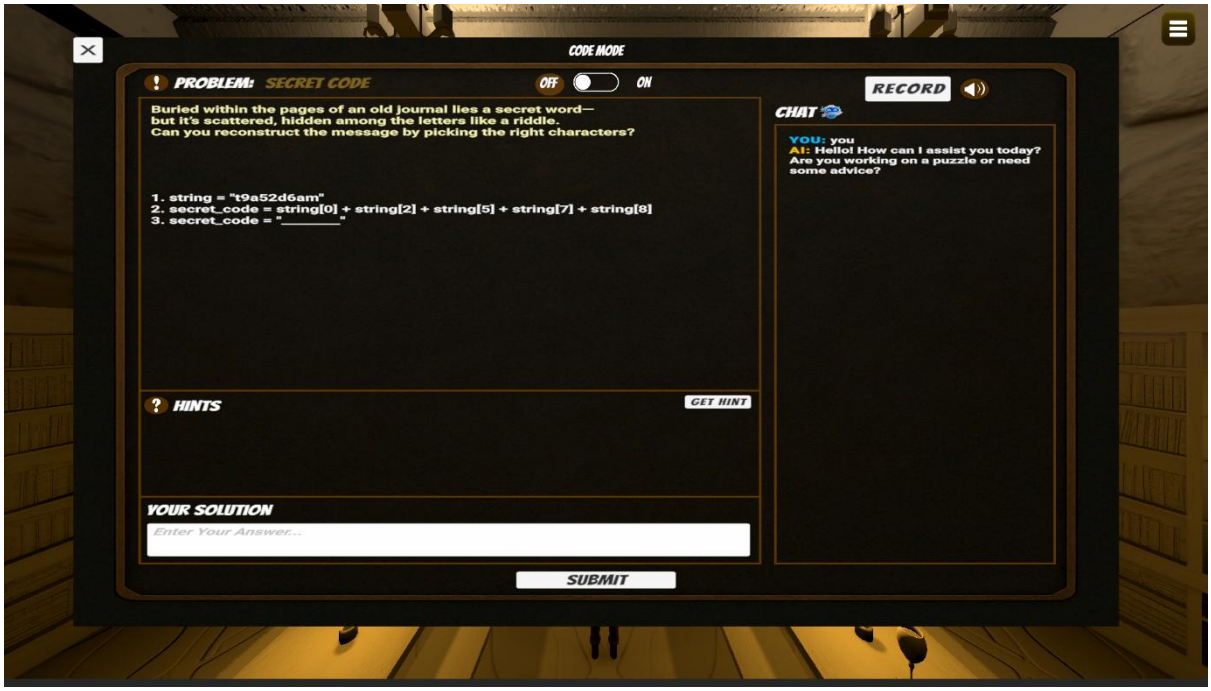
Click Get Hint if player needs a hint:



Click Record and start speaking to microphone:



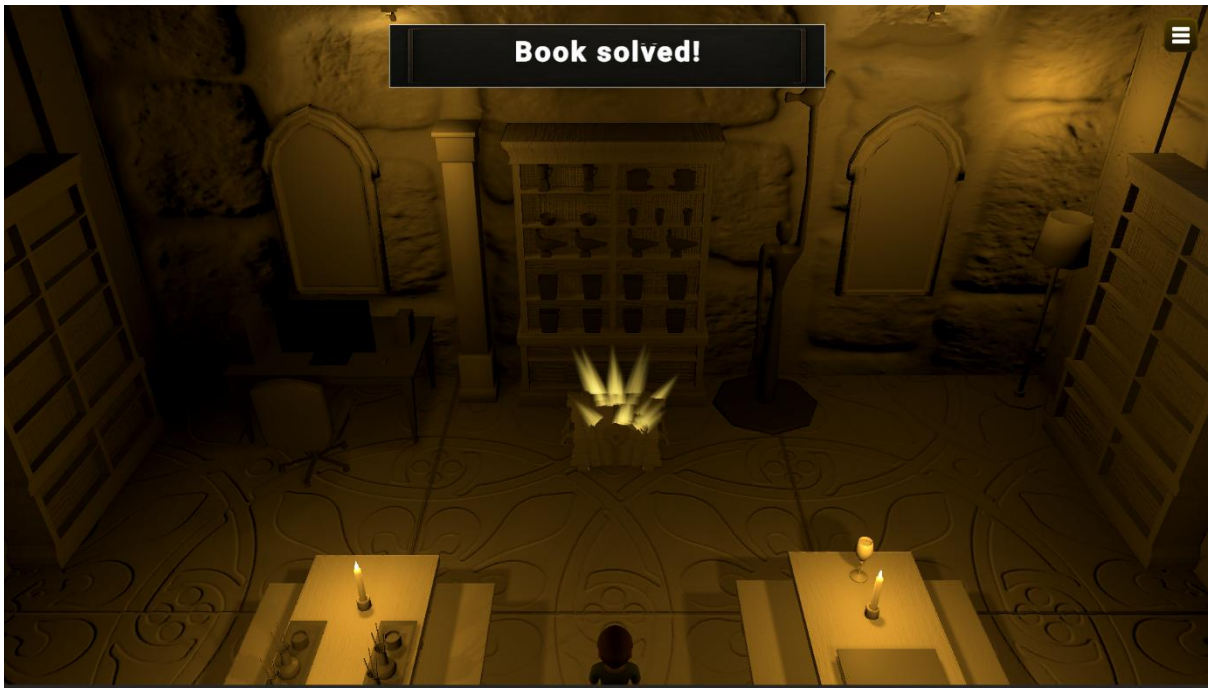
Click Stop and the AI bot will respond accordingly and help:



Click Code Mode when player has solution and wants to solve the problem:

Click Record and stop when you finish speaking. The AI bot will convert your spoken answer into a Python code structure and insert it into the 'Your Solution' tab.

Click Submit to check your solution. If it's correct, you'll proceed to the next puzzle. If not, a 'Try Again' window will appear:



11 Maintenance Guide

11.1 Software Environment and Dependencies

EscapeCode is built using the Unity game engine and integrates several external tools and APIs. Below is a list of the core software components and dependencies:

- Unity Engine (2022.3 LTS or later) – core game development platform.
- UnitEye – external black-box gaze tracking tool, modified for gray-box usage.
- Python 3.9+ – used for executing code validation scripts.
- OpenAI Whisper API – used for speech-to-text transcription.
- OpenAI GPT-4o API – provides natural language understanding and AI assistant functionality.
- Audio recording module – implemented using Unity C# and Python wrappers.
- Standard Webcam and Microphone – required hardware for gaze and voice input.

All dependencies (excluding UnitEye and OpenAI APIs) are bundled within the game structure or referenced in the repository documentation.

11.2 How to Extend / Fix / Maintain

Clone Project:

1. Clone the Repository
   - Open a terminal (or Git Bash on Windows).
   - Run the following command to clone the project to your machine: git clone https://github.com/shahaf5641/EscapeCode.git
2. Open the Project in Unity
   - Launch Unity Hub.
   - Click on "Open Project".
   - Navigate to the folder you just cloned and select it.
   - Make sure you're using Unity 2022.3 LTS or later, as required by the project.
   - If prompted to upgrade the project, click "Confirm" only if you're using the exact or newer Unity version.

3. Install Required Packages
    - Unity will automatically detect missing packages and try to resolve them via the Package Manager.
    - If something is missing:
        - Go to Window > Package Manager
        - Install the following (if not already present):
            - TextMeshPro
            - Input System (if applicable)
            - Cinemachine
            - Any custom packages (e.g., UnitEye) may need to be imported manually from .unitypackage or copied into the Assets/ folder.
4. Build Target Configuration
    - Go to File > Build Settings
    - Set Target Platform to PC, Mac & Linux Standalone
    - Choose Windows (unless another platform is intended)
5. Run the Project
    - Click the Play button in Unity Editor to test.
    - Make sure your webcam and microphone are connected if testing eye tracking or MicToVirtualClick.

<u>Project Structure Overview:</u>

The project includes multiple scenes, prefabs, and GameObjects essential for maintenance.

1. MainMenuScene
    - Main Canvas: Contains main menu buttons.
        - Level_buttons & Tutorials (Prefabs): Load different levels/tutorials.
    - MicClickManager Handles virtual clicks when eye-tracking is active.
    - GlobalInactiveObjec: Passes data between scenes. Includes:
        - Exit_Canvas.
        - CalibirtaionCanvas (for eye-tracking use).
        - Settings canvas.

    - UnitEye (eye-tracking).
- HintsGlowManager Controls hint display system.

2. FirstRoomScene
    - Player: Controls player UI and behavior.
    - Key: Integrates with the hint system.
    - Environment: Manages room visuals (add/remove/change objects).
    - CodeCanvas: Displays code interface.
    - VoiceBotManager: Integrates with Whisper and ChatGPT.

3. SecondRoomScene
    - Player: Controls player UI and behavior.
    - Free_RoomA: Manages room visuals.
    - CodeCanvas: Code interface.
    - VoiceBotManager: Speech and GPT integration.

1. TutorialScene
    - VideoManager: Plays tutorial videos.

11.3 Recommended Development Practices

- Modular Structure: Keep Unity scripts and prefabs modular and reusable. Follow the single-responsibility principle.
- Separation of Concerns: Maintain a clear separation between UI logic, game logic, and system integration (e.g., Whisper, Python).
- Testing First: Test new features in isolated Unity scenes before integrating them.
- Environment Control: Always test voice and gaze features under various lighting and noise conditions.
- API Management: Abstract API calls into helper classes to simplify upgrades or vendor changes.
- Documentation: Keep internal comments clear, maintain a README for contributors, and document API usage.
- Backup Configuration Files: Keep copies of UnitEye and Python configs before any manual modification.

12 References

1. MIT Media Lab. Scratch – Visual Programming Environment for Beginners. Retrieved from https://scratch.mit.edu
2. freeCodeCamp. Learn to Code — for Free. Retrieved from https://www.freecodecamp.org
3. CodinGame. Escape – Puzzle-Based Programming Game. Retrieved from https://www.codingame.com/start
4. Tobii AB. *Tobii eye tracking solutions*. Retrieved from https://www.tobii.com
5. Pupil Labs. *Pupil Labs – Open-source eye tracking for researchers and developers*. Retrieved from https://pupil-labs.com
6. OpenCV. *Open Source Computer Vision Library*. Retrieved from https://opencv.org
7. Emgu Corporation. *Emgu CV: Cross-platform .NET wrapper to the OpenCV library*. Retrieved from https://www.emgu.com
8. Egger, C. *GazePointer: Webcam eye tracking software*. SourceForge. Retrieved from https://sourceforge.net/projects/gazepointer/
9. OpenAI. (2023). *Whisper: Open-source speech recognition model*. Retrieved from https://github.com/openai/whisper
10. Vosk. *Offline speech recognition toolkit*. Alpha Cephei. Retrieved from https://alphacephei.com/vosk
11. Rasa. *Open Source Conversational AI*. Retrieved from https://rasa.com
12. OpenAI. (2024). *GPT-4o API Documentation*. from https://platform.openai.com/docs
13. Unity Technologies. (2024). *Unity Documentation – GameObject, Prefab and Visual Scripting*. Retrieved from https://docs.unity3d.com
14. Unity Technologies. (2024). *Unity Visual Scripting Manual*. Retrieved from https://docs.unity3d.com/Manual/VisualScripting.html
15. World Wide Web Consortium (W3C). (2018). *Web Content Accessibility Guidelines (WCAG) 2.1*. Retrieved from https://www.w3.org/TR/WCAG21/
16. Sole, G. (2023). Tiny-Whisper: Lightweight Whisper-compatible speech recognition model. GitHub. Retrieved from https://github.com/GerardSoleCa/Tiny-Whisper