BSc Computer Science for Games Programming

Honours Stage Project

Final Report

Name: Samuel Lawrence

Student Number: 202227714

Word Count: 13,834

Table of Contents

Abstract

Recent research has shown that recognising and collecting evidence at a crime scene is essential for understanding what happened and how potential offenders and victims of crime behaved. The gap in performance between experienced and novice forensic investigators highlights just how important hands-on, focused training is for developing proper search and recovery skills. Classroom teaching on its own often falls short. It's one thing to learn the theory, but another entirely to apply it under pressure in a real-world setting. That's where this project comes in. I've designed, built and tested a mixed reality game aimed at helping trainee investigators sharpen their practical skills through realistic, immersive scenarios.

The main goal was to improve how we train people for crime scene work by making it more lifelike and interactive. I used Unity to create a detailed virtual version of a training environment, and the system runs on Meta Quest 3 headsets. This setup lets trainees physically move around and interact naturally with pieces of simulated evidence. Central to my objective was to make each session feel fresh and unpredictable, so I built an AI powered solution that randomises the type, location and context of the scenarios. This pushes users to rely on meticulous observation and sound judgement, rather than memorising routines or locations often associated with predictable templated scenarios.

In short, this mixed reality tool shows how immersive tech can take forensic training to the next level. By blending realistic environments with smart, unpredictable scenarios, it offers repeated practice that is cost-effective and actually helps build confidence for fieldwork. The aim is to make the jump from simulation to real-world crime scenes as smooth and useful as possible.

(Mashable, 2023)

Acknowledgements

First off, I want to thank Dr. Xinhui Ma. Our weekly meetings gave the project its rhythm. Whether I was testing out a rough idea or stuck on a bug that refused to budge, those sessions helped me reset, rethink, and focus. Xinhui's calm questions always drew me to look at things from a sharper angle, and his feedback kept the project moving.

Massive thanks as well to Dr. Warren Viant. Whenever I encountered organisational hurdles or felt overwhelmed by the scope of my ambitions, Warren offered clear, strategic advice that helped me break down complex tasks into manageable steps.

I'm also really grateful to Dr. Darren McKie. He didn't have to get involved, but during other labs Darren was there with real world examples and solid advice. Just a couple of quick conversations with him made complex problems feel more manageable, and his suggestions genuinely changed how I approached key parts of the build.

A huge shout to Jack Thompson. Even though he's out on placement this year, Jack still found time to help me sort out the complex and confounding Quest 3 mess we encountered. I had been going in circles for weeks, and his willingness to do trial and error with me meant everything. That one breakthrough unlocked the rest of the build. Without that I would have been well and truly stuffed.

This project came together because of your support, advice, and the belief you showed me when I needed it most.

To all of you, thank you.

Introduction

Context and Background

The accurate recognition, collection, analysis, preservation and storage of evidence are foundational aspects of criminal investigations. Even the tiniest trace of DNA can end up being the key to the whole case, which means careful evidence handling isn't just recommended, it's essential. Investigators and scene support staff need sharp observation, solid procedure and the technical skills to back it up before they ever step onto a real scene. But these aren't things you can just learn from a book. They take hands-on practice, clear instruction and proper supervision to get right. A missed fingerprint, an overlooked fibre or a broken chain of custody can be all it takes to derail a case. Getting it right starts long before the first callout.

(National Center for Biotechnology Information, 2012)

Traditional training methods like classroom sessions, table-top exercises or staged mock crime scenes often fall short when it comes to capturing the full complexity of a real investigation. You can learn how to bag evidence or follow chain of custody protocols on paper, but once you're dropped into a chaotic scene with noise, distractions, time pressure and unexpected obstacles, things change. It's easy for even well-prepared trainees to miss something important or slip up on procedure. Studies show that unless people get repeated exposure to varied, realistic scenarios, their decision-making under pressure doesn't fully develop. When that happens in a real case, it can lead to missed evidence or contamination, which weakens the prosecution and sometimes even lets a suspect walk free without having been tested in the judicial system.

That's what makes mixed reality such a valuable tool. By layering virtual elements onto real physical space using a headset, it's possible to turn any room into a fully interactive crime scene. You can place virtual blood spatter, hide DNA traces, or simulate fibres and fingerprints. It all feels real, but it's controlled. Trainees can walk through the scene, interact with evidence, and get live feedback on how they're doing without risking any actual damage or compromising a real case. And of course, the budgetary and time savings are significant when compared to physical crime scene reconstruction.

For this project, I designed a mixed reality app specifically for forensic training. The concept is straightforward. Users work alone to locate and process as much evidence as they can, as accurately and efficiently as possible.

The use of game-like structure isn't just for engagement. It encourages repeated practice, improves focus, and pushes users to sharpen their skills. Every action is logged, which means feedback is specific and measurable. Both trainees and instructors can pinpoint areas of strength or spot weaknesses that need extra attention.

The overall aim is to make forensic training more practical, immersive and effective. When someone eventually enters a real crime scene, they should already have the confidence and experience to act decisively. This project is about building that foundation through repeatable, risk-free practice that speeds up learning without cutting corners.

Research question

To what extent can a mixed-reality game improve novice and even experienced forensic investigators' ability to search for, identify, recover and secure evidence in a digitally reconstructed crime scene?

Project Aims and Objectives

The core aim of the project is to build a practical, immersive training tool for trainee crime scene investigators using mixed reality. The focus is on developing confidence through repeatable, realistic scenarios that combine natural interaction, real-time feedback, and manageable complexity. Trainees should be guided through the entire evidence handling process, from initial discovery to documentation and scene completion, to using intuitive tools within dynamic environments.

The system must run natively on the Meta Quest 3, with support for both hand tracking and controllers. Real-time systems should handle scene generation, evidence detection, and performance monitoring to keep the experience accurate, responsive, and smooth.

Key features:

- Scene Generation:
    - Randomised or manual selection of Bar, Office, or Home environments
    - Probability-based object spawning, essential evidence flags, and structured presets for consistency
- Evidence Handling:
    - UV light, dust brush, and investigation camera simulate real forensic techniques
    - Evidence types include physical (e.g. weapons, phones), biological (blood, footprints), and trace (fingerprints)
- Interface & Controls:
    - World-space UI panels for tool spawning, evidence tracking, and scene control
    - Comfort features such as teleportation locomotion, pass-through integration, and scaling for accessibility
- Feedback & Replayability:
    - Live progress tracking via the Evidence Checklist
    - Progressive fingerprint revelation and automated scoring

Scope

Product Requirements

The crime scene investigation trainer is designed to run directly on the Meta Quest 3 using Unity 2022 LTS. Its main purpose is to immerse the trainee in realistic environments such as a living room, a bar, or an office, where they can move naturally, highlight items of interest, and tag evidence using controllers. The interface should feel smooth and instinctive, with panels that sit in the world or on the wrist to guide the user through each scene. To make sure every session feels fresh, forensic details like fingerprints, weapons, and blood marks will be placed differently each time the simulation starts.

I have split the build into five phases.

First, the design phase, where I studied police workflows and briefing documents. Then a proof-of-concept phase using a lightweight prototype on the Quest 3 to check interaction flow. Next comes full development, where I aim to get the entire system working as intended. After that, I will test for usability and performance, and finally, prepare everything for deployment. Throughout, I am using Unity's XR Interaction Toolkit and the Quest 3 system to keep everything reliable and responsive.

(College of Policing, 2022)

(Hornby, 2024)

Functional Requirements

The user interface for this tool will be designed around controller input, allowing trainees to grab, inspect, and tag evidence using familiar VR interactions. Movement will be supported through both teleportation and free roaming, providing flexibility and comfort during sessions. Evidence will be tagged by taking photographs with a virtual camera, and all menus will be positioned in world space, floating at comfortable, accessible positions to ensure quick and intuitive access throughout the simulation.

On the functional side, the tool needs to load and display different scene types, randomise the placement of forensic traces every time it runs, and let users interact with the environment such as: photographing evidence, revealing fingerprints etc. There will also be simple controls to reset the scene or move on to the next one. To keep everything responsive, I am targeting a steady frame rate of at least 60 frames per second, and scene load times under five seconds. For safety, the system will always check the headset's guardian boundaries and pause or recenter the scene automatically if the user drifts too far. Comfort settings like vignette control and snap turn will be included for users who prefer them. To make the experience reliable, hand tracking will fall back to controllers if needed. No personal data will be collected so no need to track or manage data under GDPR. All logs will be saved locally and kept anonymous.

Finally, I will be working within the hardware constraints of the standalone Quest 3, which means keeping the memory use under one gigabyte, keeping shaders lightweight, and making sure collision checks don't exceed two milliseconds per frame. All of this will help the system stay smooth and stable within the headset's battery and performance limits.

Design Constraints

All development will be built in Unity 2022 LTS using the XR Interaction Toolkit alongside Meta Quest 3's features. Every scenario and forensic narrative will be influenced by real police briefing documents to make sure the procedures reflect actual fieldwork. The entire experience is designed with hand tracking in mind, but controllers are the main source of input. The randomisation system that places forensic traces must avoid overlapping objects and always keep items within easy reach of the trainee. Language used throughout the tool including labels, instructions, and evidence descriptions, will aim to follow guidelines set by the College of Policing, in UK English, especially around how evidence is handled and how chain of custody is recorded.

(College of Policing, 2022)

Risk Analysis

Technological Risks

Virtual reality hardware and software has become much more reliable in recent years, but it still isn't immune to problems. Headsets can crash without warning, sensors can lose track of movement, batteries run flat mid-session, and unexpected bugs can throw off the whole experience. When that happens during a training scenario, it breaks immersion and makes it harder to debrief properly afterwards. Over time, it can also damage confidence in the system. To prevent this, the equipment needs to be looked after properly. That means checking for software updates every day, running full diagnostics on the headsets every week, and monitoring everything while it's running. Charging stations should show battery levels clearly, so no one starts a session with a headset that's going to die. Session data should be backed up constantly in the background so that nothing gets lost. If something does go wrong during a session, the trainee can be quickly switched to a clean, ready-to-use headset, with minimal disruption.

(Road to VR, 2024)

Skill Transfer Risk

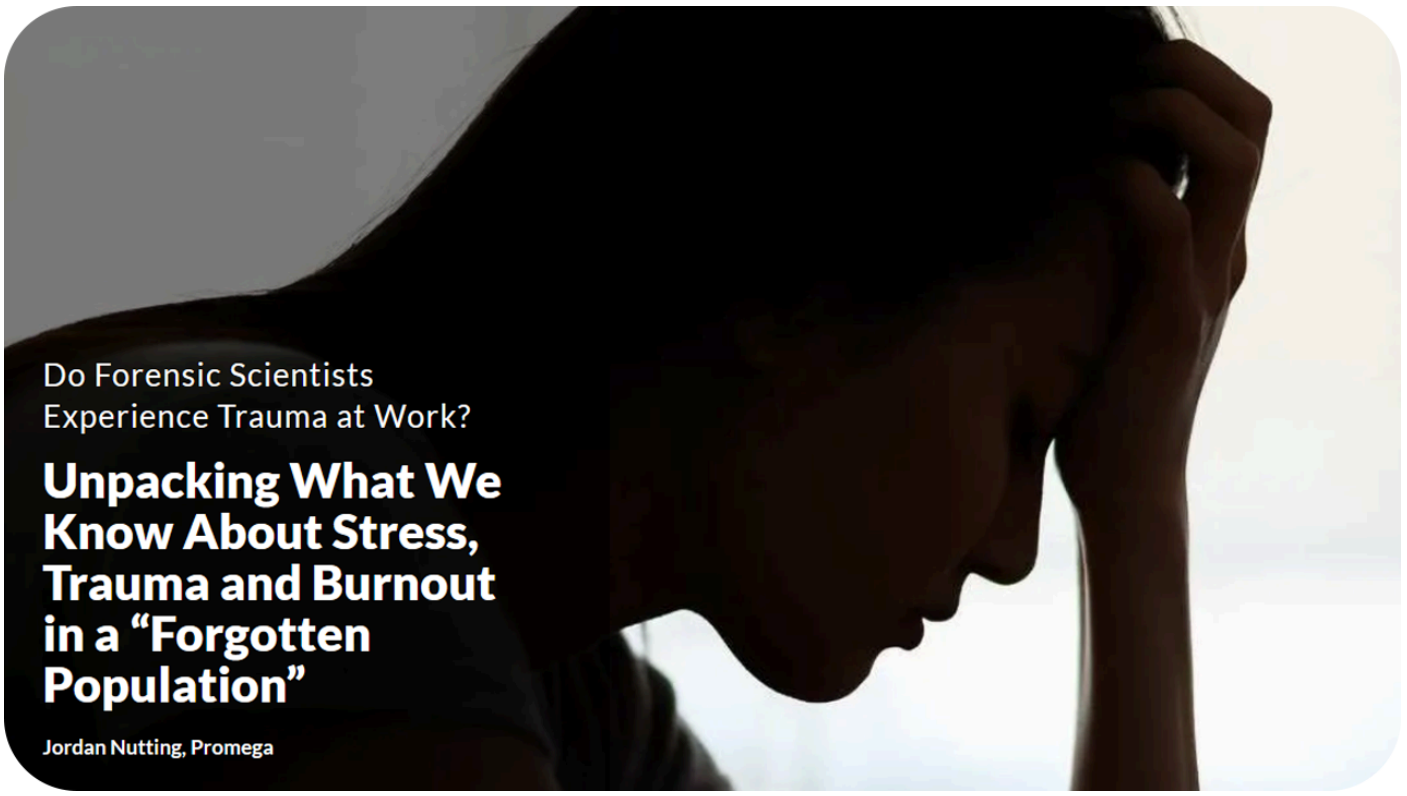
Even the most detailed simulation isn't a perfect replacement for the real world. No matter how good the graphics are or how sharp the sounds feel, a virtual crime scene won't have the same distractions, sensations, emotions or unpredictability as an actual scene. That can be a problem if trainees come out of VR feeling confident, but then freeze or miss something important when they're faced with the real thing. To narrow the gap, virtual scenarios need to feel more like the live environments they're preparing for. That means adding things like poor lighting and noise in the background. Each virtual session should also be paired with a practical follow-up, where trainees work with real props and mocked-up scenes to reinforce what they've learned. Progress should be tracked before and after each scenario, and instructors should make sure to explain how the simulation connects to the real decisions they'll face later on.

Time Overrun

VR development can often take longer than you estimate. Between modelling the assets, programming the interactions and testing the whole thing properly, the hours quickly add up. If things slip, it causes delays in testing, frustrates the people waiting to use it, and in the real word of work puts pressure on budgets. In this academic case however, it could mean an unfinished project when it comes to the hand-in date. To avoid this, the work should be broken down into clear stages, with each one focused on a specific target. Progress should be reviewed every two weeks preferably with academic monitors who can provide independent and more objective input. That way, risks are spotted early and dealt with before they affect the final delivery.

Emotional Distress

Some trainees may find certain scenarios distressing. Even when they know it's not real, the sight of graphic or distressing environments can potentially trigger strong or unexpected reactions. In the worst case scenario, it could lead to someone dropping out of the training altogether. To prevent this, everyone taking part should complete a short mental health check beforehand and give their informed consent. At the end of each session, trainees should be encouraged to reflect and talk through what they experienced. If anyone has a strong negative reaction, the session is paused, and they're given immediate support.



(Promega, 2023)

Literature Review

Crime Scene Investigation Training

Crime scene investigation training has always relied on three main pillars: real world placements, classroom teaching, and practical simulations. Together, these build the core of what makes a capable forensic investigator. Out of all of them, nothing beats a proper placement. Standing at a real scene, working alongside seasoned officers, is where theory gets stress tested by reality. That moment when you step past the tape is when everything you thought you knew suddenly matters in a new way. Light changes by the minute. Buildings might be falling apart. Rain or heat threatens to wash away or distort evidence. And the clock is always ticking. Trainees have to act fast, whether it is lifting prints before the surface bakes dry, or sealing up key items before contamination sets in. Even small tasks like filling out paperwork become critical when you are trying to focus while the scene is active.

But placements are not equal. One person might get sent to a serious case with firearms and multiple victims, while another gets routine call outs for break ins or vandalism. Access to high risk scenes is often limited, so the most valuable learning chances are sometimes the ones no one gets. And although being at a real scene helps cement knowledge like nothing else, it can also leave a mark. Without proper check ins and emotional support, the stress and trauma can build. Some trainees never really talk about what they have seen. Few training programmes build in formal debriefs or give space for mental health to be taken seriously.

Classroom time lays the groundwork. This is where students are taught the science behind different types of evidence, the laws that shape the process, and the ethical standards that are expected at every stage. The topics range widely: blood patterns, DNA traces, ballistics, legal rulings, and expert testimony all get covered. There is also focus on how bias creeps in, and what happens when a process is broken or a rule ignored. It is vital knowledge, but it only takes you so far. No exam prepares you for when evidence goes missing or a scene turns out to be nothing like the one described. Being able to pass a theory test does not guarantee someone will cope under real-world pressure.

That is where simulations can contribute. They provide a middle ground, letting trainees practise in a safe and repeatable way. These might include actors playing witnesses, carefully staged rooms full of prop evidence, or mocked up environments that use lights, smells, and sounds to build a bit of tension. Instructors can run the scene multiple times, step in with pointers, and check that everything is being done properly. For early training, these are great. But they have their own limits. Once a trainee has done the same scenario a few times, they start to remember where things are. Instead of searching or thinking, they are repeating. That turns training into a memory test, not a problem solving exercise. Most setups still fall short of showing the full impact of a real crime scene. The weight of it, the pressure in the air, the unexpected reactions. Those are still hard to fake, and often missing from even the best designed simulations.

Mixed Reality in Law Enforcement Training

Mixed reality and virtual reality platforms offer a promising way to close some of the persistent training gaps that classroom instruction and physical simulations cannot fully bridge. These technologies immerse trainees in detailed digital environments that can be reset instantly, controlled precisely, and adjusted to introduce new challenges as needed. With MR headsets like the Meta Quest or Microsoft HoloLens, an ordinary room can be transformed into a fully interactive crime scene.

The real power of mixed reality lies in how it combines physical motion with digital elements. Unlike a fully virtual environment that relies on joysticks or teleportation controls, MR allows users to walk naturally, lean in to inspect clues, and interact with their surroundings in intuitive ways. Trainers can adjust the conditions in real time, switching to low light to simulate a night time search, triggering background noise to test focus, or introducing new pieces of evidence mid scene. Because the entire environment is virtual, there is no risk of biohazard, contamination, or property damage. And since scenes can be generated procedurally or altered randomly, trainees cannot fall into the trap of memorising a simulation.

(Apex Officer, 2023)

AI in Training and Evidence Randomisation

In modern forensic training, a lot of systems now use AI to keep exercises from feeling too predictable. Instead of running the same scene over and over, they'll shuffle the layout, change where objects are placed, or rotate through a few preset witness statements. If a trainee misses something important, the system might flash up a hint or reveal a backup clue. It keeps things from going stale, but in most cases the core story doesn't change. The clues don't connect in any meaningful way. You end up with randomness for the sake of it, rather than something that actually feels like a proper investigation.

Real evidence randomisation should go much further. Starting with just a few environments: a flat, an office, or maybe a bar, the system should rebuild everything from scratch. It doesn't just move things around. It reshapes the story behind the scene. That broken window might shift from the front to the side to make it look like a forced entry. A bloodied glove might be found half hidden in a drawer, suggesting a rushed cleanup. And instead of standalone clues, things like footprints, fibres, and witness statements should all line up to tell the same story, or deliberately mislead in a consistent way. When this kind of system works properly, every session feels fresh but still believable. It forces trainees to think about motive, means, and opportunity, not just what's lying around. It adjusts as they go, tracking what they've seen, how long they looked at something, or what they missed. That makes each run feel like a real case. It's not just about spotting clues. It's about figuring out how they fit together, what they mean, and what to do next. This is the fictional pinnacle in my opinion for where Mixed Reality CSI could reach.

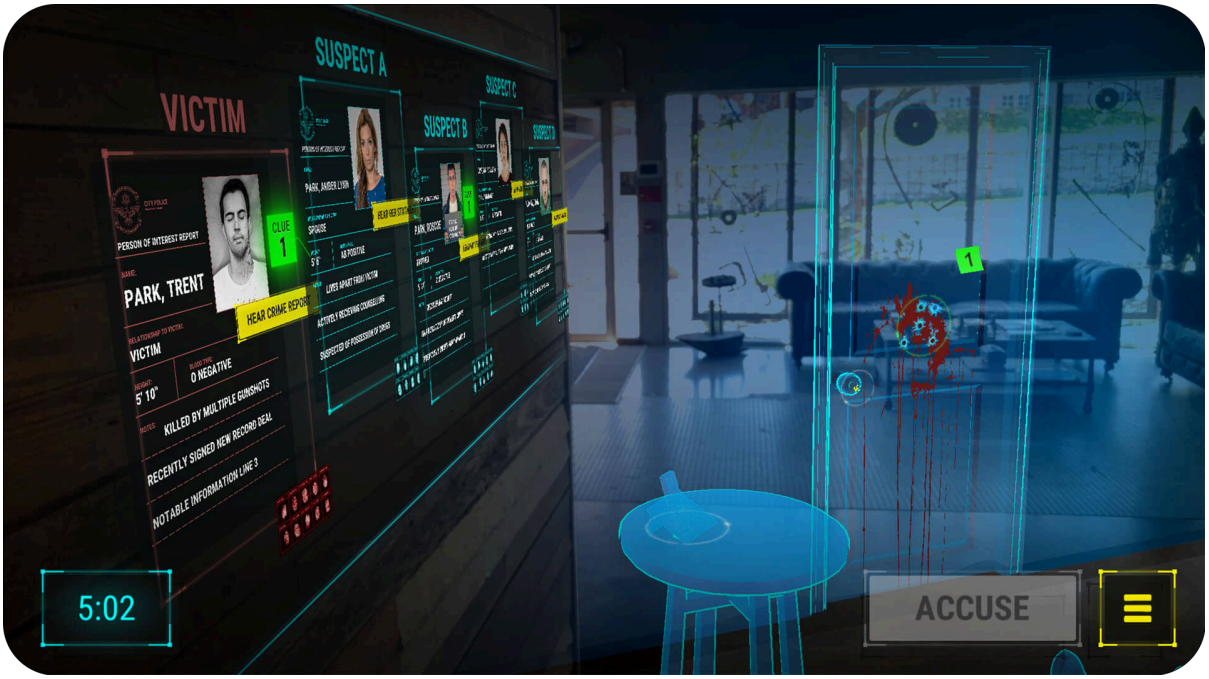
(IMDb, n.d.)

Existing Tools and Technologies

Many of the current training platforms that use Mixed Reality or artificial intelligence are built using the Unity game engine. Unity is popular in this space because it offers strong physics simulation, high quality visuals, and a wide range of plugins and development tools. Its physics engine can simulate things like the way blood travels from an impact, or how bullets behave when they hit barriers (too high level for this project), and it handles materials like shattered glass or torn clothing with impressive realism. Developers can use Unity's High Definition Render Pipeline to adjust lighting and shadows so that a crime scene feels like it takes place under the right conditions, whether it is the soft glow of an indoor lamp or the harsh glare of outdoor sun.

But the process of building a single training scenario in Unity is still slow and complex. It starts with 3D modelling teams who design each prop and environment, from pieces of furniture to entire buildings. Programmers then add interactivity and write scripts for how the AI responds. Forensic experts are brought in to check the accuracy of every element, while audio designers layer in sounds like traffic, weather, or crowd noise to give the scene a realistic atmosphere. These teams often work in separate software tools like Blender or Maya to create assets, which are then imported into Unity and manually connected to the AI systems through custom bridges. That kind of pipeline takes time and coordination.

There is also the cost. Unity's professional versions are not free, and neither is the hardware needed to run high quality simulations. Most systems require powerful computers, specialist headsets, and sometimes additional gear like haptic gloves for realistic interaction. For smaller police departments or universities, these costs can quickly become a barrier. Without outside funding or dedicated technical teams, many organisations simply do not have the resources to use these tools at scale. As a result, adoption is often limited to larger institutions that can afford both the upfront investment and the ongoing support. To streamline this for the current project, it's possible to use standalone XR hardware like the Meta Quest 3. By targeting Quest 3's optimized Universal Render Pipeline, you can run high-fidelity scenes without the need for a tethered PC or expensive GPU, reducing both equipment and licensing costs. Hand-tracking and built-in haptics eliminate the need for specialist gloves or external sensors, while on-device rendering and wireless streaming keep the setup simple.



(NeoPangea, n.d.)

Ethical and Legal Considerations

Ethical and legal considerations are paramount when developing a training tool that could be used by law enforcement agencies. The following issues have been planned for below:

- **Privacy concerns**: The game tracks trainee performance and collects personal data. To comply with GDPR, data will be anonymized, securely stored. Users should be informed beforehand.
- **Legal considerations**: All external datasets or materials must be properly licensed for educational or commercial use.
- **Emotional distress**: Given the crime scene simulations, care must be taken to provide psychological support for trainees who may be affected by these scenarios (For client to organise).

Commercial Considerations

While this project is primarily educational, it has potential commercial value:

- **Development Costs**: The total cost of development is projected at £300, covering only the hardware costs.
- **Market Potential**: Once developed, the game will be offered to the Humberside police and hopefully some academic institutions. It could also be adapted for use in civilian training, such as crime scene investigator courses or watered down for just an experience/game.
- **Competition**: There are existing crime scene simulation tools, but some, if any, integrate mixed-reality technology in the way my project aims to do. This gives the game a competitive edge in the training market.

Gaps in the Literature

Even with all the attention mixed reality and artificial intelligence are getting, in forensic training the research still has a long way to go. Most studies stop at the surface. They measure things like whether a trainee enjoyed the tool, or how fast they finished a task. But they rarely look at what actually matters, whether the training sticks, whether people remember what they learned months later, or whether it improves how they perform in real investigations.

There's also very little work on how forensic knowledge is actually built into AI systems. It's one thing to shuffle objects around, but if the clues don't follow proper logic, the whole scenario falls apart. Most systems don't check whether the evidence makes sense as a complete case. And hardly anyone is comparing different AI models to see which ones hold up under different conditions.

The same gaps show up on the ethical side. A lot of papers mention stress, or the risk of emotional overload, but very few go further. There's not much testing on whether debriefs or mental health check-ins actually help. When it comes to cost, people note that these tools are expensive or hard to scale, but there's little detail on how to make them cheaper, easier to build, or shareable between institutions. Open-source pipelines and collaborative libraries are barely mentioned.

(Reid and Moray, 2024)

Technical Achievement

System Design and Architecture

Overall System Overview

The architecture of XM 24 856 is built with modularity, performance, and long term scalability in mind, specifically for standalone mixed reality hardware like the Meta Quest 3. I structured the system into distinct layers, each with a clear job. At the base is the platform layer, made up of the Quest hardware and Unity's XR subsystems. Above that is the engine layer, handling core services and rendering. Sitting on top are the core systems such as scene generation, evidence tracking, and narrative logic. The interaction layer includes all player tools and input handling, while the presentation layer covers the UI and audio feedback.

Everything communicates through an event based model. Instead of hard coding dependencies, systems subscribe to lifecycle events such as when scene generation finishes. This approach keeps things loosely connected and easier to maintain. I can change or extend features without breaking unrelated parts of the system, which makes testing and updates much safer.

The application lifecycle is also clearly defined. When the app starts, services register themselves using a lightweight locator pattern. The Crime Scene Generator builds the scene and places all evidence. Once that's done, it sends out a ready signal. From there, UI components and player tools initialise based on the current state of the scene. At the end of each session, whether the checklist is complete or the user restarts, all temporary data is cleared and the system resets for a new run.

I use Scriptable Objects extensively to store configuration: spawn rules, location weighting, tool settings, colours, and scenario presets. Keeping data separate from logic makes it easy to tweak behaviour quickly without digging into code.

Everything is optimised for the limitations of the Quest 3. I maintain performance through batching, object pooling, and coroutine based updates. Unity events keep the communication between systems quick and lightweight, helping to meet frame rate targets without relying on tethered computing or high end hardware.

Core Features

The project's functionality is built around three key systems: the Crime Scene Generator, the Evidence Checklist, and the Tool Spawner. These systems work in sync to create dynamic forensic environments, track user progress, and enable lifelike interactions throughout the simulation.

The Crime Scene Generator builds each training scenario from scratch. It randomly selects one of three templates between Bar, Office, or Home, and activates the corresponding environment while disabling the others to reduce clutter. Within the selected location, evidence is placed at designated spawn points, filtered by surface type such as floor, table, or wall. Each item uses weighted probabilities to determine if it should appear, ensuring that no two scenes feel the same while always including essential items needed for learning outcomes. To help scenes feel more natural and less scripted, each item is given a small random rotation on placement. For formal assessments or guided lessons, structured presets can be used to control exactly which items appear and where.

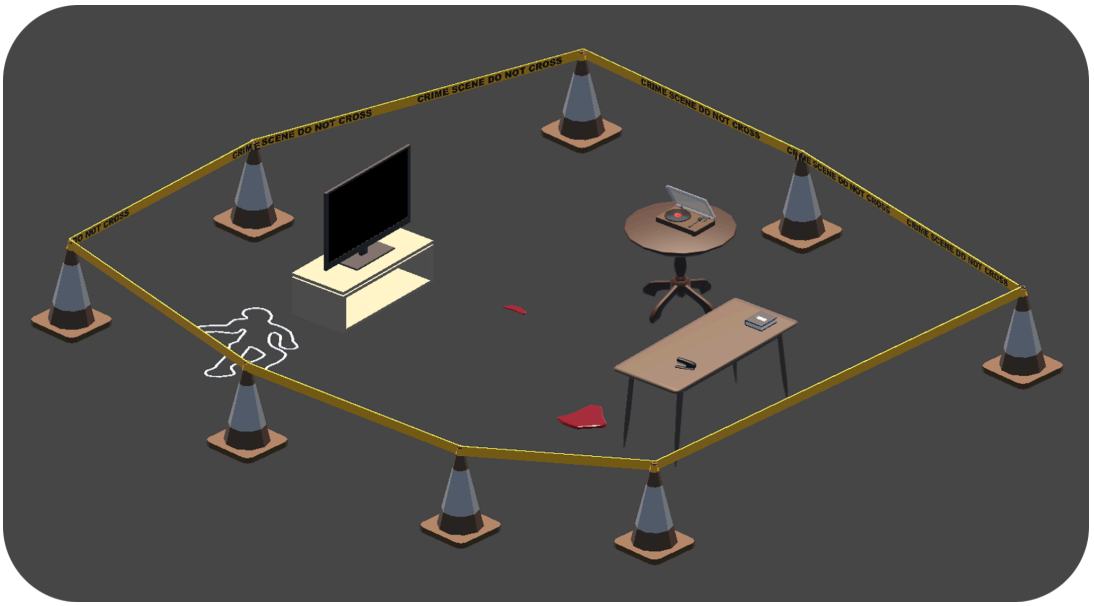
The Evidence Checklist tracks all evidence throughout the simulation. It keeps a live record of each object's visibility, discovery status, and whether it has been photographed. This system drives the interface that trainees use to monitor their progress. When an item is discovered or captured, the checklist updates instantly, helping trainees see how far they've come and what still needs to be done. It connects directly with the camera and fingerprint systems, ensuring smooth integration and accurate scoring.

Trainees interact with the environment using tools provided by the Tool Spawner. This system offers access to core forensic instruments like the UV torch, dust brush, evidence scanner, and digital camera. Each tool behaves differently and simulates real forensic processes. The UV torch makes hidden fingerprints glow under ultraviolet light. The dust brush reveals latent prints through soft visual transitions. The camera uses a wide ray grid to detect evidence in a natural, seamless way that doesn't interrupt the viewfinder or make the user feel disconnected.

Navigation is handled by a teleportation system designed with user comfort in mind. Teleport points appear only when needed and fade out when they're not in use, keeping the interface clean and easy to read. The system automatically adapts to different user heights and play areas, allowing for smooth, comfortable movement throughout the scene.

The simulation is built around a tight feedback loop. Every time the trainee finds or logs evidence, the checklist updates instantly, which in turn updates the UI. This reinforces learning in real time, encourages detailed scene analysis, and helps build investigative thinking. It also ensures that progress tracking is accurate from start to finish, making it easy to assess trainee performance at a glance.


CRIME SCENE DO NOT CROSS


CRIME SCENE DO NOT CROSS


CRIME SCENE DO NOT CROSS

User Interface (UI) Design

The user interface is built entirely in world space and designed specifically with the Quest 3's comfort and clarity in mind. Each UI panel is scaled and positioned to sit naturally within the trainee's field of view, either hovering at a comfortable distance or anchored to the wrist for quick access without breaking immersion.

The core interface is the Evidence Checklist. It updates automatically after each scene is generated, listing all relevant items and tracking whether they've been found or photographed. Each entry uses colour and icons to show status clearly. An optional "mystery mode" hides item names until they're discovered, adding an extra layer of challenge. The checklist also is easily toggled to reduce distraction.

Tool management is handled through the Tool Spawner Interface. This menu is split into two tabs, one for spawning tools like the UV torch and camera, and another for scene controls like resets or toggling spawn behaviour. For testing, there are debug-only sliders that adjust how much evidence appears or how likely certain items are to spawn.

Throughout the simulation, audio and visual cues help guide the user. Sound effects confirm actions like checklist updates, button presses, or tool activations. Animations draw attention to milestones and make feedback feel more polished and engaging.

To support comfort and accessibility, the interface includes adjustable scale settings, snap turns, and smooth teleportation transitions. The app also monitors the Quest 3's boundary system and automatically pauses if the user steps out of range.

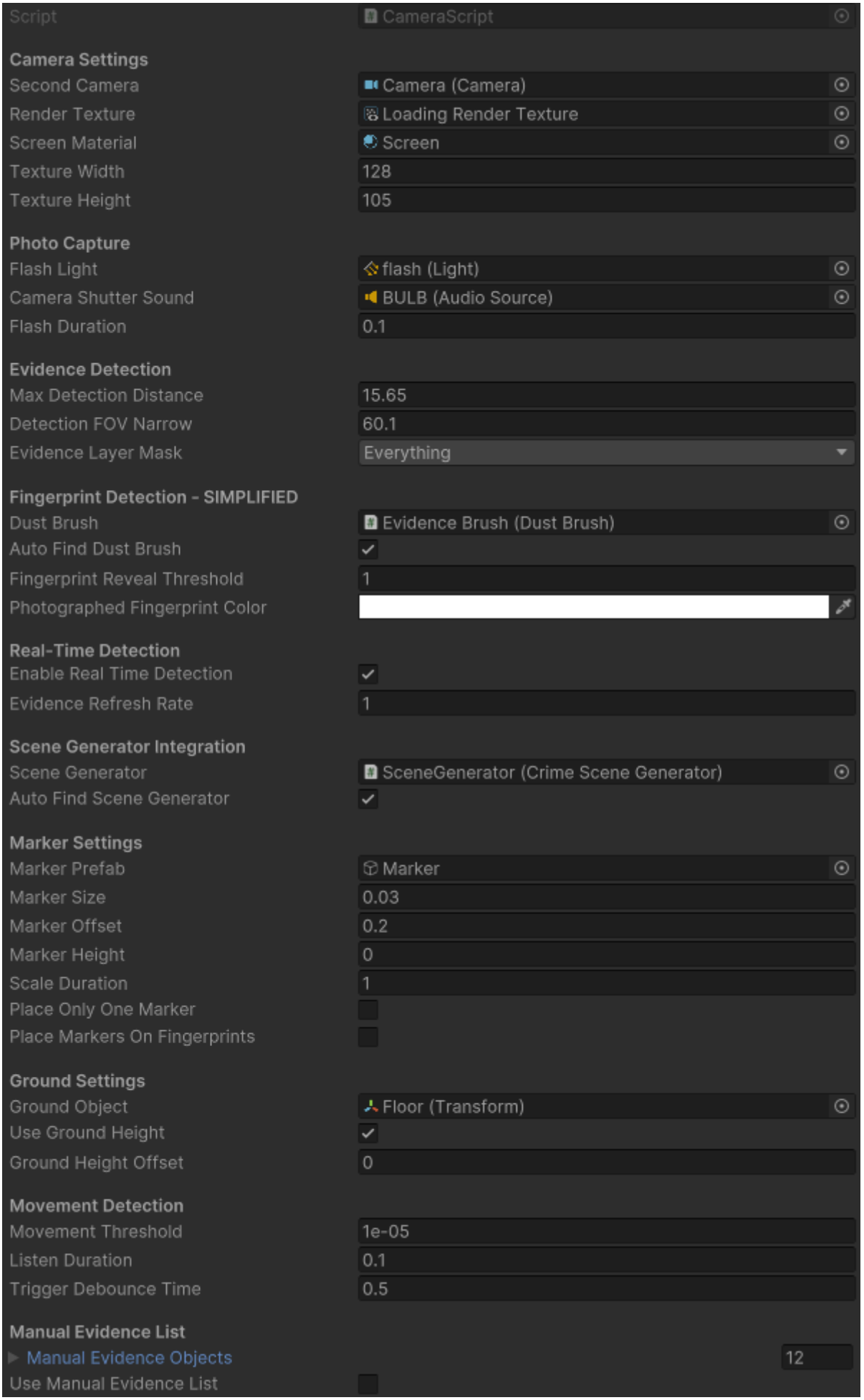
Altogether, the UI strikes a balance between usability and immersion. It provides the information trainees need without clutter, supports multiple control methods, and reinforces a smooth, responsive training experience.

Script Overview and Feature Descriptions

Camera Script (CameraScript.cs)

The Camera Script controls how the mixed reality camera behaves during photo capture. It sets up a secondary render texture, handles smooth effects like flash and shutter sounds, and introduces dynamic field-of-view expansion when scanning for evidence. This is achieved by casting a dense grid of rays with a temporarily widened angle, so even off-centre or edge evidence is picked up, without disrupting the user's visual perspective. The result is an intuitive capture process that feels natural.

```
// Much denser ray grid for maximum sensitivity
int raysPerAxis = 7;
float fovMultiplier = 3.0f;
float expandedFOV = detectionFOVNarrow * fovMultiplier;

// Main grid pattern
for (int x = 0; x < raysPerAxis; x++)
{
  for (int y = 0; y < raysPerAxis; y++)
  {
    // Skip center ray (already cast)
    if (x == raysPerAxis / 2 && y == raysPerAxis / 2) continue;

    // Calculate viewport coordinates with much wider spread
    float normalizedX = (float)x / (raysPerAxis - 1);
    float normalizedY = (float)y / (raysPerAxis - 1);

    // Expand from center with the wider FOV
    float viewportX = 0.5f + (normalizedX - 0.5f) * (expandedFOV / 60f);
    float viewportY = 0.5f + (normalizedY - 0.5f) * (expandedFOV / 60f);

    Ray ray = secondCamera.ViewportPointToRay(new Vector3(viewportX, viewportY, 0));
    RaycastHit[] hits = Physics.RaycastAll(ray, maxDetectionDistance, evidenceLayerMask);
    allHits.AddRange(hits);
  }
}
```


Script
CameraScript
Camera Settings
Second Camera
Camera (Camera)
Render Texture
Loading Render Texture
Screen Material
Screen
Texture Width
128
Texture Height
105
Photo Capture
Flash Light
flash (Light)
Camera Shutter Sound
BULB (Audio Source)
Flash Duration
0.1
Evidence Detection
Max Detection Distance
15.65
Detection FOV Narrow
60.1
Evidence Layer Mask
Everything
Fingerprint Detection - SIMPLIFIED
Dust Brush
Evidence Brush (Dust Brush)
Auto Find Dust Brush
Fingerprint Reveal Threshold
1
Photographed Fingerprint Color
Real-Time Detection
Enable Real Time Detection
Evidence Refresh Rate
1
Scene Generator Integration
Scene Generator
SceneGenerator (Crime Scene Generator)
Auto Find Scene Generator
Marker Settings
Marker Prefab
Marker
Marker Size
0.03
Marker Offset
0.2
Marker Height
0
Scale Duration
1
Place Only One Marker
Place Markers On Fingerprints
Ground Settings
Ground Object
Floor (Transform)
Use Ground Height
Ground Height Offset
0
Movement Detection
Movement Threshold
1e-05
Listen Duration
0.1
Trigger Debounce Time
0.5
Manual Evidence List
Manual Evidence Objects
12
Use Manual Evidence List

Crime Scene Generator (CrimeSceneGenerator.cs)

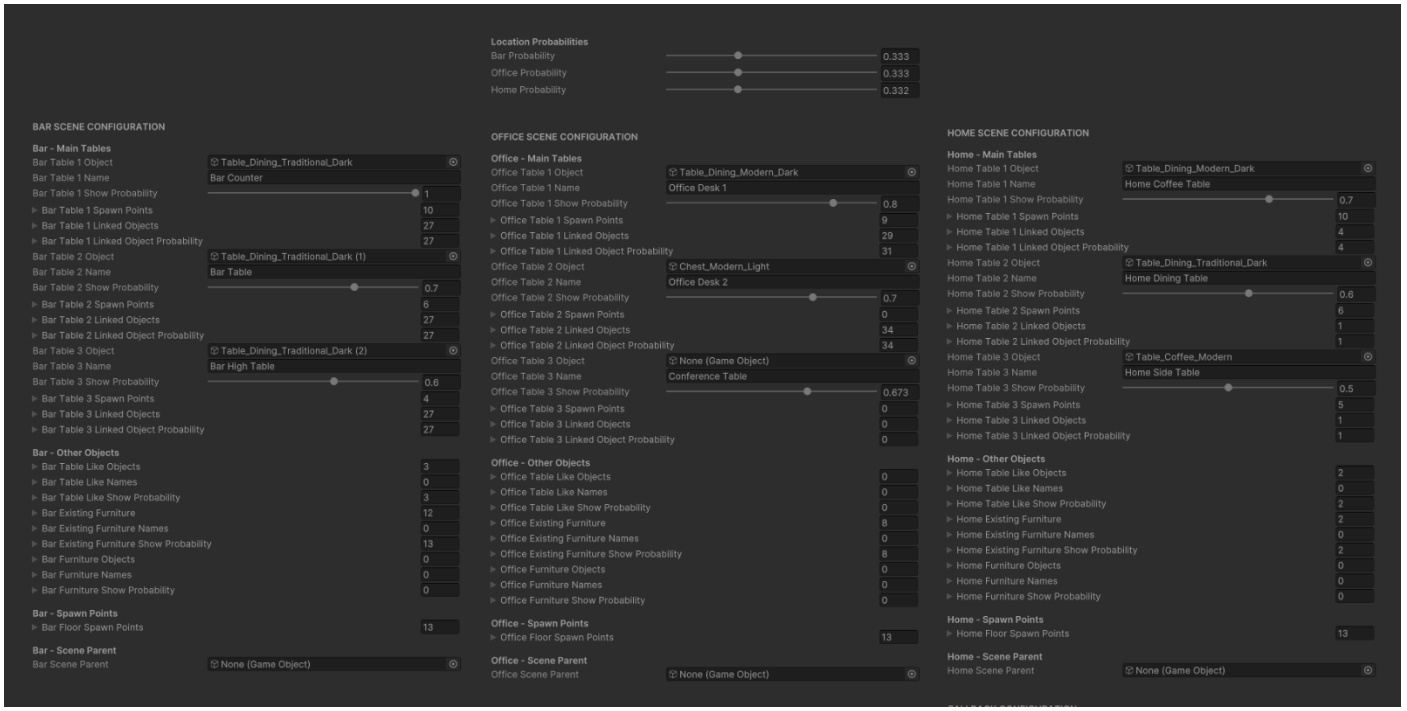
The Crime Scene Generator is the core system that builds each training scenario. It automatically fills environments with furniture, props, and forensic evidence, using either fully random layouts or structured presets. This allows each run to feel fresh and unpredictable, while still teaching key investigative principles. Its procedural setup means trainers don't have to manually design every scene, making it ideal for replayable, varied, and realistic CSI training sessions.

Scene Types and Location Logic

The generator supports four distinct environments: Bar, Office, and Home. Each one includes unique spawn points, furniture layouts, and object groupings tailored to that setting. When generating a scene, the system assigns a scene type using weighted random selection, unless a specific environment is manually chosen. This ensures varied layouts while still allowing targeted testing or demonstrations.

```
// - SCENE LOCATION DECISION
private void DecideSceneLocation()
{
  if (!enableLocationSelection)
  {
    currentLocation = CurrentLocation.Fallback;
    isLocationDecided = true;
    return;
  }

  if (useRandomSelection)
  {
    float randomValue = (float)randomizer.NextDouble();
    ...
  }
  else
  {
    switch (forceLocationIndex)
    {
    ...
    }
  }

  isLocationDecided = true;
}
```

This setup keeps each run fresh. When a location is chosen, its parent GameObject (for example barSceneParent) is enabled and all others are turned off to keep the hierarchy tidy and avoid distraction. The HandleLocationSpecificObjects() method handles that switching.



Furniture and Table Logic

Each scene includes several table objects, each with their own props like glasses, books, or phones etc, and a series of potential spawn points. Their visibility is determined using assigned probabilities in the Unity Inspector. The method below evaluates whether each table and its contents should appear. This ensures a different combination of objects each time, giving the environment a more natural, lived-in feel.

```
// - TABLE PROCESSING
private void ProcessLocationTable(GameObject tableObject, string tableName, float showProbability,
            Transform[] spawnPoints, GameObject[] linkedObjects, float[] linkedObjectProbability)
{
  if (tableObject == null) return;

  bool shouldShow = randomizer.NextDouble() < showProbability;
  tableObject.SetActive(shouldShow);

  if (shouldShow)
  {
    ...
    if (spawnPoints != null)
    {
      spawnedTables.AddRange(spawnPoints);
    }
  }
  else
  {
    HideLinkedObjects(linkedObjects);
  }
}
```

By selectively showing or hiding props using probabilistic logic, the simulation avoids repetition and feels more natural. This principle is also applied to standalone props using a similar approach through the ProcessSimpleObjects() method.

Evidence Placement

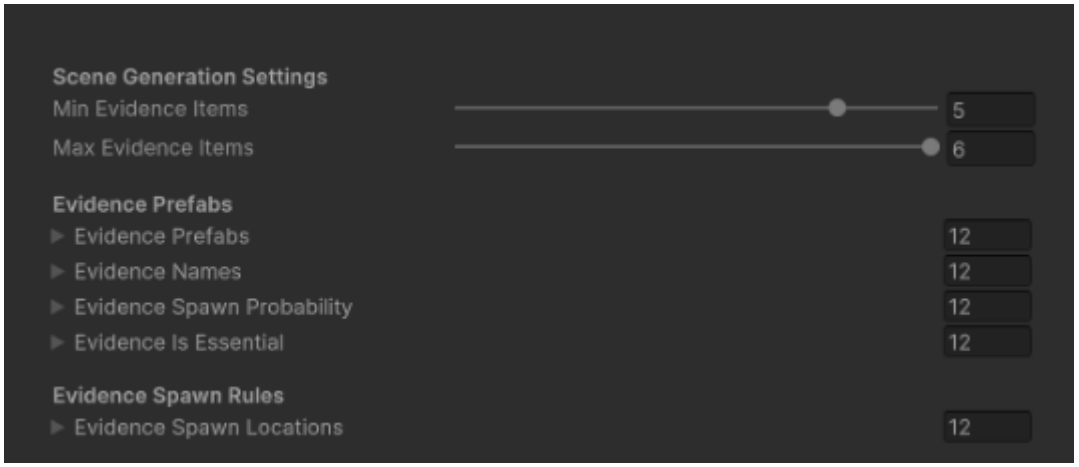
Evidence items are placed in the scene randomly. Each item is configured with a spawn type, such as floor only, table only, or either; a probability of appearing; and a flag that marks whether it is essential to the investigation. To determine the correct spawn point, the system filters through all available locations using tags that indicate whether the point is on the floor or on a table. The method below shows how that filtering works in practice. This ensures each piece of evidence appears in a plausible and accessible location. Essential evidence is always spawned, while non-essential items are subject to randomness, which helps maintain a sense of realism and replayability across scenarios.

```
// - SPAWN POINT SELECTION
Transform[] GetAppropriateSpawnPoints(int evidenceIndex)
{
  List<Transform> validPoints = new List<Transform>();
  SpawnLocation location = evidenceSpawnLocations[evidenceIndex];

  if...
```

The generator then instantiates each evidence prefab at a valid spawn point, rotating it randomly around the Y-axis to avoid patterns that could make the simulation feel artificial or predictable. This subtle variation in orientation helps reinforce the impression of natural clutter or realistic object placement, especially when revisiting the same environment multiple times.

```
// - EVIDENCE SPAWNING
void SpawnEvidence(int evidenceIndex)
{
  if (evidenceIndex >= evidencePrefabs.Length || evidencePrefabs[evidenceIndex] == null)
    return;

  Transform[] spawnPoints = GetAppropriateSpawnPoints(evidenceIndex);

  Vector3 targetPosition;
  Transform selectedSpawnPoint = null;

  if (spawnPoints.Length == 0)
  {
    return;
  }
```

The use of essential flags in each evidence object guarantees that critical items will appear no matter what their associated probability value is. This ensures that key learning outcomes, such as identifying fingerprints on a murder weapon or locating pools of blood, are always available during training, even as the rest of the scene changes around them.



Scenario Presets

In educational use, consistency is sometimes just as important as variability. I started working on supporting repeatable sessions for assessment or instructional purposes, so the generator could also allow evidence to be spawned according to predefined scenario presets done by the developer. However, I turned my attention over to Version 1.0 which I talk about later in this report.

```
// - PRESET GENERATION
void GenerateFromPreset()
{
  if (selectedScenarioIndex >= scenarioPresets.Length)
    selectedScenarioIndex = 0;

  int[] preset = scenarioPresets[selectedScenarioIndex];

  foreach (int evidenceIndex in preset)
  {
    if (evidenceIndex < evidencePrefabs.Length && evidencePrefabs[evidenceIndex] != null)
    {
      SpawnEvidence(evidenceIndex);
    }
  }
}
```

Real-Time Integration

Once a scene has been generated, the generator triggers updates to external systems such as the camera script and the evidence checklist. These updates are sent using coroutine-delayed events to allow Unity's rendering cycle to complete before dependent systems begin their own initialisation routines.

```
// - SYSTEM NOTIFICATION
private IEnumerator NotifySystemsAfterGeneration()
{
  yield return new WaitForEndOfFrame();
  yield return new WaitForSeconds(notificationDelay);

  if (notifyCameraOnGeneration && cameraScript != null)
  {
    string[] evidenceNamesArray = currentSceneEvidenceNames.ToArray();
    cameraScript.UpdateEvidenceList(evidenceNamesArray);
    cameraScript.OnSceneGenerated();
  }

  if (notifyChecklistOnGeneration && evidenceChecklist != null)
  {
    evidenceChecklist.OnSceneGenerated();
  }
}
```

This timing ensures that the in-simulation camera and the evidence UI are fully synchronised with the current scene layout. For example, the evidence checklist will accurately reflect the number and type of items placed, and the camera will have correct references for framing and detection zones. By coordinating these systems in real time, the simulation maintains consistency and avoids early-access errors or stale data.

Cleanup and Regeneration

Before generating a new scene, the system clears out all previously spawned objects to avoid clutter and ensure accuracy. Any existing evidence prefabs are removed, fingerprints are reset to their initial hidden state, and environmental markers are cleared. This guarantees that each session starts from a clean slate and that trainees interact only with the intended scenario layout.

```
// - SCENE CLEARING
[ContextMenu("Clear Scene")]
public void ClearScene()
{
  // Reset evidence tracking
  currentSceneEvidenceNames.Clear();
  currentSceneEvidenceObjects.Clear();

  // Destroy all spawned objects
  if (spawnedObjects != null)
  {
    foreach...
    }
    spawnedObjects.Clear();
  }
```

In summary, the Crime Scene Generator serves as the backbone of environmental variation within the MR simulation. Its ability to combine procedural content with predefined structure allows for both unpredictability and control. Whether used for assessment, repetition, or guided learning, it ensures every run feels fresh while remaining grounded in authentic investigative logic. Its seamless communication with other systems and flexible configuration options make it a vital component of the CSI training pipeline.

Dust Brush (DustBrush.cs)

The Dust Brush script brings fingerprint dusting to life by simulating how investigators reveal prints during a real crime scene search. It plays a key role in building immersion, letting trainees interact directly with surfaces and uncover hidden evidence in a realistic way. When the brush makes contact with a fingerprint, the script kicks into action, either instantly revealing the print in test mode or gradually during standard use. The effect builds up as the trainee continues brushing, with visual feedback showing the fingerprint becoming clearer over time. This responsive interaction makes it easy to tell when the brush is working, helping reinforce good technique and attention to detail.

```
// - FINGERPRINT REVEAL SYSTEM
// Gradually reveal fingerprint over time
void RevealFingerprint(GameObject fingerprint)
{
  if (!revealProgress.ContainsKey(fingerprint))
  {
    revealProgress[fingerprint] = 0f;
  }

  // Increase reveal progress
  float currentProgress = revealProgress[fingerprint];
  float newProgress = Mathf.MoveTowards(currentProgress, 1.0f, revealSpeed * Time.deltaTime);
  revealProgress[fingerprint] = newProgress;

  // Swap to revealed material when fully revealed
  if (newProgress >= 1.0f && !fullyRevealedFingerprints.Contains(fingerprint))
  {
    SwapToRevealedMaterial(fingerprint);
    fullyRevealedFingerprints.Add(fingerprint);
  }
}
```

Evidence Checklist (EvidenceChecklist.cs)

The Evidence Checklist manages real time tracking and display of all key items within the crime scene simulation. It gives trainees a clear, constantly updated view of their progress by listing each discovered object and photographed fingerprint in a popup menu panel.

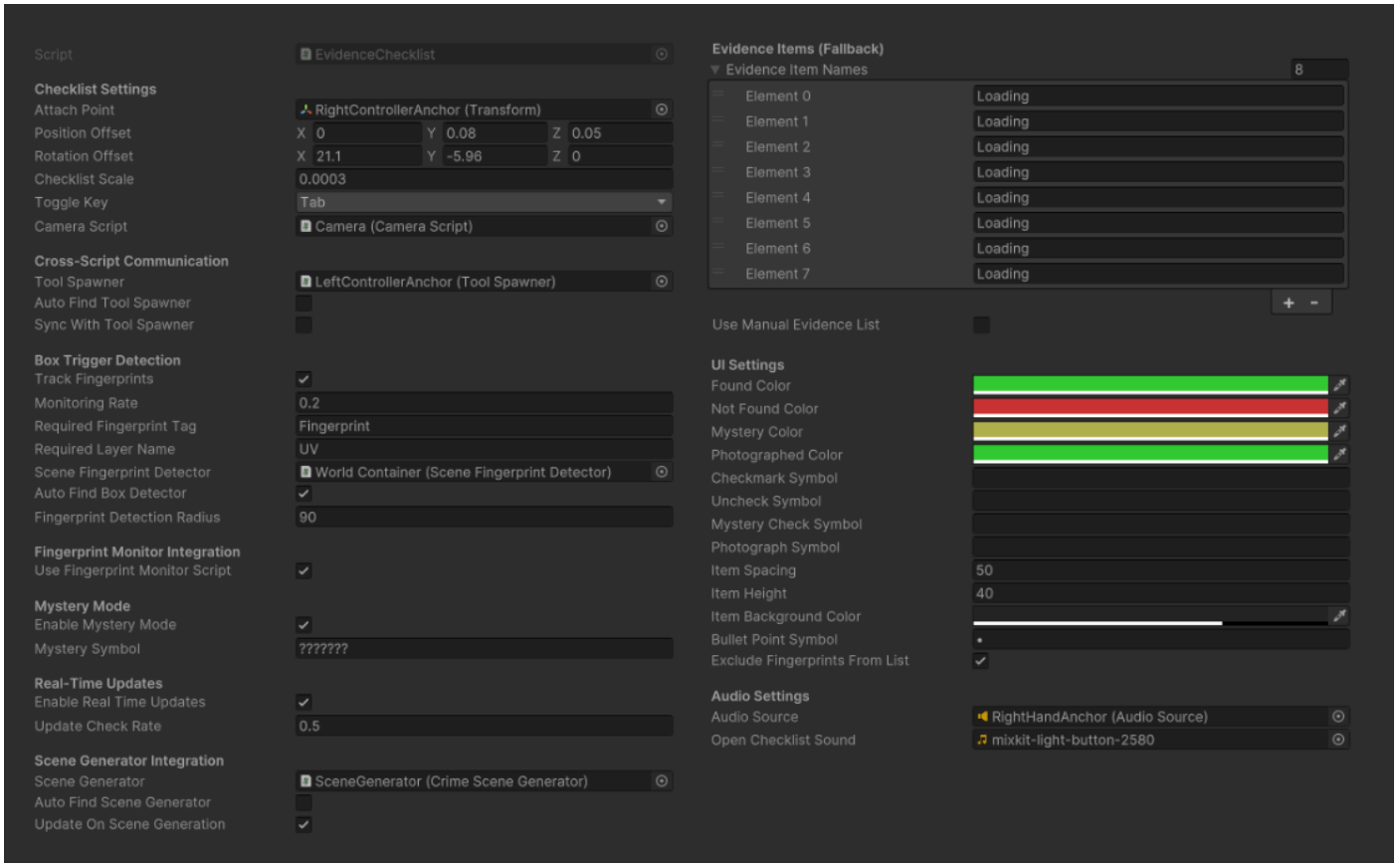
When a new scene is generated, the checklist populates automatically by pulling data from the Crime Scene Generator or a manually defined list. Fingerprints are tracked using their unique instance IDs and given clear names for easy identification.

```
// - FINGERPRINT IDENTIFICATION
private string GetUniqueFingerprintId(GameObject fingerprint)
{
  // Generate unique ID for fingerprint
  if (fingerprint == null) return "NullFingerprint";
  return $"Fingerprint_{fingerprint.name}_{fingerprint.GetInstanceID()}";
}
```

Each entry shows a label, a colour-coded progress marker, and icons to indicate whether the item has been found or documented. The whole checklist can be toggled open or closed using the controller input.

```
// - CHECKLIST VISIBILITY TOGGLE
public void ToggleChecklist()
{
  // Toggle checklist visibility
  isChecklistVisible = !isChecklistVisible;
  if (checklistInstance != null)
  {
    checklistInstance.SetActive(isChecklistVisible);
    if (isChecklistVisible)
    {
      UpdateChecklistTransform();
      UpdateChecklistUI();
      if (audioSource != null && openChecklistSound != null)
      {
        audioSource.PlayOneShot(openChecklistSound);
      }
    }
  }
}
```

Designed to be lightweight and responsive, this system provides instant feedback, helps prevent missed items, and encourages users to fully investigate each scene before wrapping up.



Fingerprint Monitor (FingerprintMonitor.cs)

The Fingerprint Monitor script tracks whether fingerprint objects are visible to the main camera and if they've been brushed enough to count as revealed. It runs lightweight checks at regular intervals, confirming visibility and querying the Dust Brush system, to support things like score updates, UI feedback, or training prompts. It uses simple visibility tests without heavy processing, keeping the simulation responsive while quietly monitoring trainee progress.

```
// - STATUS UPDATE SYSTEM
// Update and process fingerprint status
private void UpdateStatus()
{
  bool isVisible = IsVisibleToCamera();
  bool isBrushed = IsBrushed();
}
```

Fingerprint Detection (SceneFingerprintDetection.cs)

The Fingerprint Detection system scans the environment to identify valid fingerprint objects and updates the evidence checklist once they are discovered. It plays a vital role in making sure all fingerprint evidence is recognised, logged, and visually tracked throughout the simulation.

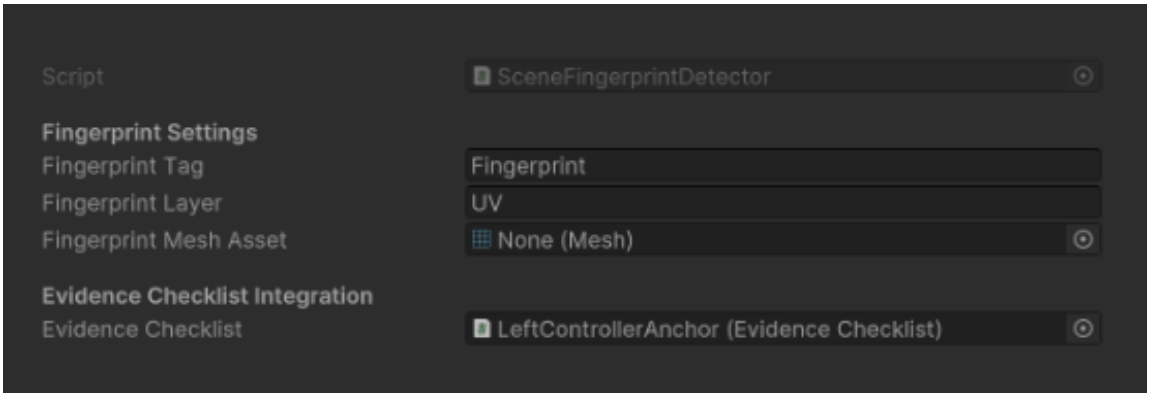
Each potential fingerprint is checked using a series of strict validation rules. The system examines the object's tag, layer, mesh type, renderer, and naming convention. These checks are all handled by the IsFingerprint() method.

```
// - FINGERPRINT VALIDATION
private bool IsFingerprint(GameObject obj)
{
  // Check if object meets fingerprint criteria
  if (obj == null) return false;

  // Ensure the object is active in the hierarchy
  if (!obj.activeInHierarchy) return false;

  // Check for MeshRenderer component
  MeshRenderer meshRenderer = obj.GetComponent<MeshRenderer>();
  if (meshRenderer == null) return false;

  // Check for correct mesh
  MeshFilter meshFilter = obj.GetComponent<MeshFilter>();
  if (meshFilter == null || meshFilter.sharedMesh == null) return false;

  // Compare the sharedMesh with the serialized fingerprintMeshAsset
  if (fingerprintMeshAsset == null || meshFilter.sharedMesh != fingerprintMeshAsset)
  {
    return false;
  }
```

When DetectAllFingerprintsInScene() is called, the system loops through every active GameObject, runs the fingerprint validation, and adds any confirmed prints to an internal list. These are then passed directly to the Evidence Checklist so they show up in the trainee's UI.

```
foreach (GameObject obj in allActiveGameObjects)
{
  // Only process active game objects
  if (obj != null && obj.activeInHierarchy && IsFingerprint(obj))
  {
    detectedFingerprints.Add(obj);

    // Notify evidence checklist if available
    if (evidenceChecklist != null)
    {
      evidenceChecklist.OnFingerprintEnteredBox(obj);
    }
  }
}
```

This detection can run automatically at the start of each scene, or be triggered manually later if needed. This flexibility ensures that fingerprints are always correctly tracked, whether the layout was randomly generated or built from a fixed scenario.



Menu Actions (MenuActions.cs)

The Menu Actions script handles all menu-based operations for the crime scene training tool. It controls safe scene transitions, quitting the application and audio feedback. When a user starts the investigation scene, it plays a sound, then immediately generates the scene. The scripts all work together to clear anything left over from the last scene. This prevents unwanted behaviour like ghost inputs or stuck UI elements.

To quit the application, the script performs a similar cleanup process before exiting. If it is running in the Unity Editor, it stops play mode rather than closing the whole editor. There is also a built-in fallback in case a scene load or quit command fails the first time.

```
public void StartInvestigation()
{
  if (isTransitioning)
  {
    return;
  }

  PlayButtonSound();
  StartCoroutine(SafeSceneTransition(investigationSceneName));
}
```

A key feature is its VR cleanup system. This shuts down any active XR or Oculus components, disables interactive scripts such as the teleport system or tool spawner, and clears out any lingering static references. This helps avoid bugs during scene reloads or repeated testing.

```
// - SCENE TRANSITION SYSTEM
// Safe scene transition with VR cleanup
private IEnumerator SafeSceneTransition(string sceneName)
{
  isTransitioning = true;

  // Play transition audio
  PlayTransitionSound();

  // Perform VR cleanup if enabled
  if (enableVRCleanup)
  {
    yield return StartCoroutine(PerformVRCleanup());
  }

  // Standard transition delay
  yield return new WaitForSeconds(transitionDelay);

  // Load scene with error handling
  LoadSceneWithFallback(sceneName);
}
```

For sound design, it includes optional clips for button clicks and scene transitions, adding polish to interactions. Developers can manually trigger VR cleanup for debugging, and all cleanup methods use Unity's coroutine system to avoid blocking or hard crashes.

In short, this script ensures that all menu actions are performed cleanly and predictably, making it a vital part of maintaining stability during both development and training sessions.

Teleport System (TeleportSystem.cs)

The Teleport System makes it easy for users to move around the virtual space, helping them navigate comfortably and stay focused during training. It lets trainees jump between fixed locations without breaking immersion, making it easier to explore detailed scenes and stay engaged.

Key features include:

- Controller-based movement: Users trigger teleportation with the standard VR controller, offering a reliable and familiar way to get around.
- Smooth camera transitions: The system eases the user between positions with a gentle motion, which helps avoid motion sickness and keeps things comfortable.

Behind the scenes, the system scans for valid teleport points at startup and keeps track of them throughout the session. It also updates the beam effect in real time, so users can aim precisely before they move.

```
// - TP POINT INITIALIZATION
// Initialize all teleport points in scene
void InitializeTPPoints()
{
  tpPoints.Clear();

  // Get TP points from parent if available
  if (tpPointsParent != null)
  {
    GetTPPointsFromParent();
  }
  else
  {
    // Find TP points in scene
    FindTPPointsInScene();
  }

  // Initialize all found TP points
  foreach (TPPoint point in tpPoints)
  {
    point.Initialize(alphaThresholdProperty);
  }
}
```

```
// Update beam visual appearance
private void UpdateBeamVisual(Vector3 origin, Vector3 direction, Quaternion rotation)
{
  if (beamSegmentObjects == null || beamSegmentObjects.Length == 0)
    return;

  float beamLength = raycastDistance;

  // Position and orient each beam segment
  for (int i = 0; i < beamSegmentObjects.Length; i++)
  {
    GameObject segment = beamSegmentObjects[i];
    if (segment == null) continue;

    // Calculate segment position
    float segStart = (float)i / beamSegmentObjects.Length;
    float segEnd = (float)(i + 1) / beamSegmentObjects.Length;
    float segCenter = (segStart + segEnd) / 2;
```

Teleportation is executed smoothly, maintaining user comfort and preserving immersion within the training scenarios.

```
// Execute the actual teleportation
  void ExecuteTeleportation()
  {
    Vector3 targetPosition = currentTargetPoint.transform.position;
    targetPosition.y = transform.position.y; // Maintain player height

    // Handle held objects
    List<Transform> heldObjects = new List<Transform>();
    List<Vector3> relativePositions = new List<Vector3>();

    if (teleportHeldObjects)
    {
      FindHeldObjects(heldObjects);

      // Store relative positions
      foreach (Transform heldObj in heldObjects)
      {
        relativePositions.Add(heldObj.position - transform.position);
      }
    }

    // Teleport player
    transform.position = targetPosition;

    // Move held objects
    for (int i = 0; i < heldObjects.Count; i++)
    {
      if (heldObjects[i] != null)
      {
        heldObjects[i].position = transform.position + relativePositions[i];
      }
    }
  }
```

This approach ensures effective, accessible, and comfortable navigation within the virtual training environment.

Tool Spawner (ToolSpawner.cs)

The Tool Spawner script controls how trainees access and use forensic tools inside the simulation. It manages everything from spawning each tool to making sure it behaves the right way once it's in the trainee's hand. At the start of a session, the script loads each item like the dusting brush, UV torch, evidence scanner, or investigation camera, and quietly moves them out of view into a designated storage zone. This keeps the workspace clear until the user needs something. When a tool is selected through the menu, it spawns with its correct size, physics settings, and interaction logic ready to go.

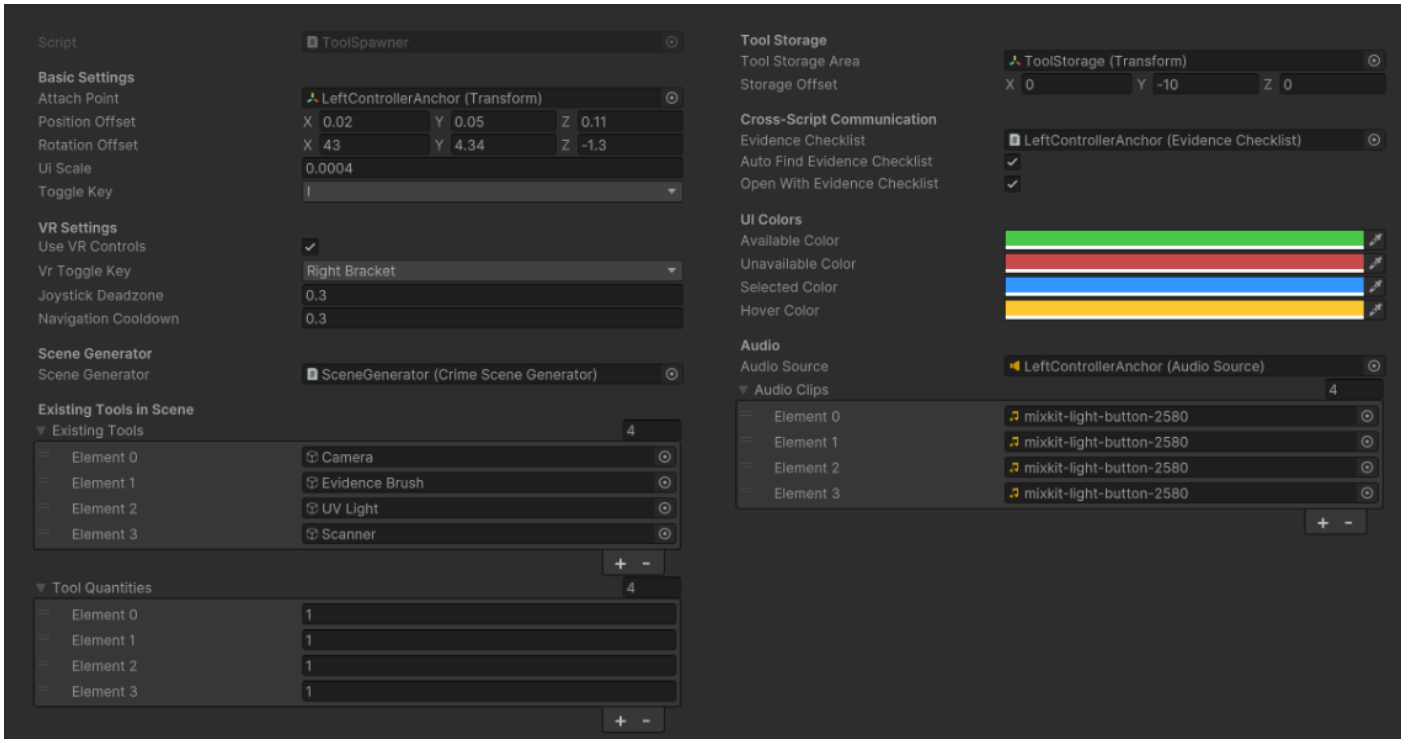
Each tool has its features built in. The UV torch for example reacts to certain trace types. To make it feel intuitive, the script tracks where the player's hand is, anchors the interface close by, and automatically despawns any unused tools after a set time. This prevents clutter and ensures the experience stays smooth.

```
// - TOOL INITIALIZATION
void InitializeExistingTools()
{
  // Setup each existing tool in the scene
  for (int i = 0; i < existingTools.Length; i++)
  {
    if (existingTools[i] != null)
    {
      // Store original position and move to storage
      originalToolPositions[i] = existingTools[i].transform.position;
      toolsActive[i] = false;
      MoveToolToStorage(i);
    }
    else
    {
      // Try to find tools by name if not assigned
      GameObject foundTool = GameObject.Find(toolNames[i]);
      if (foundTool != null)
      {
        existingTools[i] = foundTool;
        originalToolPositions[i] = foundTool.transform.position;
        toolsActive[i] = false;
        MoveToolToStorage(i);
      }
    }
  }
}
```

The script allows tools to be shown or hidden as needed, automatically moving them from the storage area to the trainee's hand when selected through the interface. This makes switching between tools fast, smooth, and easy to understand, especially during time-sensitive tasks.

```
void MoveToolToPlayer(int toolIndex)
{
  // Show tool and move to player
  if (toolIndex >= 0 && toolIndex < existingTools.Length && existingTools[toolIndex] != null)
  {
    Vector3 spawnPos = Camera.main.transform.position + Camera.main.transform.forward * 0.5f;
    existingTools[toolIndex].transform.position = spawnPos;
    existingTools[toolIndex].SetActive(true);
    toolsActive[toolIndex] = true;
  }
}
```

Interactions work naturally across VR controls, using Meta Quest 3 standard controllers. Real time updates to the interface, smooth navigation, and clear audio cues all help make the experience intuitive. Trainees can move through tasks quickly and with confidence, without needing to fight the controls.



Teleportation Point (TPPoint.cs)

The Teleport Point script manages how each teleport location appears within the scene. It allows points to smoothly fade in or out. Using methods like RevealPoint(duration) and HidePoint(duration), the script animates material transparency over time, providing a more polished and intuitive user experience. These transitions help guide trainees visually, if they choose to do so, while keeping the interface clean and immersive.

```
// Animate alpha threshold over time
IEnumerator AnimateAlphaThreshold(float targetThreshold, float duration)
{
  // Skip animation if no materials available
  if (materials.Count == 0)
  {
    currentAnimation = null;
    yield break;
  }

  // Get starting threshold from first material
  float startThreshold = materials[0].GetFloat(alphaThresholdProperty);
  float time = 0;

  // Animate over duration
  while (time < duration)
```

UV Light (UVLight.cs)

The UV Light script allows trainees to simulate ultraviolet lighting techniques used in real investigations, helping reveal fingerprints that would otherwise stay hidden. When activated, the handheld spotlight reacts to the environment in real time. If a fingerprint falls within the cone of the light, it begins to glow. As the light moves away, the fingerprint gradually fades back out. This effect is handled frame by frame, giving a smooth transition between visible and invisible states.

The script runs a check on all fingerprint objects in the scene. If one is inside the beam, the ApplyUVEffect method adjusts its transparency and glow settings. If the fingerprint moves out of range, the RestoreOriginalMaterial method resets its colour, texture, and emission back to normal. The spotlight is fully interactive, controlled through the Meta Quest 3 controllers, so users can examine different surfaces closely and with precision.

This system adds an extra layer of realism to the training tool. It challenges users to think like real forensic investigators, scanning the scene carefully and learning to pick up on subtle details. It also helps reinforce good observation habits by making the discovery process feel earned and satisfying.

```
// Apply UV glow effect to fingerprint
void ApplyUVEffect(Renderer renderer)
{
  // Set target opacity to revealed level
  targetOpacity[renderer] = uvRevealedOpacity;

  // Apply glow emission
  if (renderer.material.HasProperty("_EmissionColor"))
  {
    renderer.material.SetColor("_EmissionColor", glowColor);
    renderer.material.EnableKeyword("_EMISSION");
  }

  // Configure transparency for URP
  ConfigureTransparency(renderer);
}
```

Implementation and Development

Development Tools

I built the application in Unity, targeting the Meta Quest 3 as a fully standalone mixed reality platform. For interaction, I relied on Unity's XR Interaction Toolkit, which let me implement gesture-based controls and six-degrees-of-freedom head tracking without needing to write any custom tracking logic. This streamlined the development process and kept everything compatible with Unity's built-in input system.

The Unity Physics engine powered all collision-driven interactions, like the brush strokes used to uncover fingerprints. I used the Animator system to deliver smooth, responsive feedback, both in the UI and in tool-specific actions. Every menu flicker, object spawn, or checklist update was backed by animations that made the experience feel polished and reactive.

I chose the Universal Render Pipeline (URP) to keep performance high on the Quest 3. URP allowed me to tune lighting and shader complexity while still maintaining visual clarity. To keep everything modular and flexible, I leaned heavily on Unity's prefab system. This let me reuse assets cleanly and rapidly build different training scenes without breaking anything. The Unity Asset Store was also helpful for placeholder models and early UI elements, especially during the proof-of-concept phase.

For version control, I used GitHub throughout development, with Unity Collaborate briefly used during prototyping for device compatibility tests. I performed real-time debugging via OVR Metrics Tool, logging directly from the headset during live sessions to track down issues in rendering, tool logic, or UI responsiveness.

(Meta, n.d.)

AI Algorithms

Evidence Randomisation

Evidence placement in the simulation is handled by a custom probability-weighted system. Each scene includes a configurable list of evidence prefabs, with each item assigned a spawn probability and a designated surface type, floor, table, or both. When a new scene is generated, the system begins by selecting an environment type (Bar, Office, or Home), either randomly or from a predefined choice. It then activates the correct parent object for that location and deactivates the others to keep the scene clean and focused.

Next, it gathers all available spawn points, filtering them based on surface type and whether they are currently visible. Once the valid spawn points have been identified, the system evaluates each evidence prefab against its probability setting. Those that pass the check are randomly assigned to one of the suitable spawn locations and instantiated into the scene. This process creates unique combinations each run, while ensuring that every piece of evidence appears in a realistic and accessible location.

Pseudocode

The shuffle logic introduces variation with every session, making sure that no two runs feel exactly the same. At the same time, the “essential” flag acts as a safeguard to ensure that core training objectives, such as locating key fingerprints or identifying primary weapons, are always met. This balance of randomness and control maintains both engagement and educational consistency. Linked objects, such as props related to the evidence, are toggled using their own spawn probabilities. This not only enhances realism but also deepens the narrative density of each scene, encouraging trainees to piece together the broader story behind the evidence.

```
Function GenerateScene():
    If randomizer is null:
        create new randomizer

    Reset all fingerprints

    If location selection is enabled:
        DecideSceneLocation()

    If clearPreviousScene is true:
        ClearScene()

    HideAllEvidencePrefabs()
    Clear evidence tracking lists

    HandleLocationSpecificObjects()

    If using scenario presets:
        GenerateFromPreset()
    Else:
        GenerateRandomScene()

    If real-time notifications are enabled:
        Start notification coroutine
```

Hardware Integration

To integrate the application with the Meta Quest 3, I used Unity's XR Plugin Management system paired with the OpenXR backend. This setup gave me access to Quest-specific features like hand tracking, passthrough support, and hardware optimised rendering. The XR Interaction Toolkit was essential for treating the user's hands as fully tracked input devices, enabling natural interactions such as picking up tools, dusting for fingerprints, or scanning evidence without needing physical controllers.

The interaction model was built around the user's physical presence. UI panels automatically positioned themselves within comfortable arm's reach and rotated to follow the user's gaze, maintaining visibility without feeling intrusive. Evidence items only became interactive when the player was close and had a direct line of sight, which encouraged real-world movement and deliberate inspection, just like in an actual investigation.

To improve user comfort, especially during longer training sessions, I avoided artificial locomotion altogether. Instead, I relied on room-scale movement and teleportation to reduce motion sickness. I also implemented dynamic UI scaling that adjusted interface size based on user distance, keeping everything legible without overwhelming the field of view. Haptic pulses and spatial audio cues provided an extra layer of feedback when tools interacted with tagged surfaces, reinforcing correct behaviour through physical and sensory confirmation.

Unit Testing

Isolated Unit Testing

To make sure each core system worked correctly on its own, I wrote and ran unit tests covering scene logic, evidence tracking, and basic user interface behaviour. These tests focused on checking expected outcomes, tracking changes in system state, and verifying how random elements behaved under different conditions. For example, I tested that the EvidenceChecklist properly updated discovery status for each object, and that essential evidence always appeared, no matter which environment was selected. This helped catch logical issues early on, especially in systems that were built to run independently without relying on other parts of the application.

Runtime Testing with OVR Metrics Tool

Beyond regular testing, I used the OVR Metrics Tool to monitor live performance directly on the Meta Quest 3 headset. It gave me real-time access to essential data like frame timing, processor load, temperature levels, and memory usage. This meant I could debug the application during actual use while handling tools, moving through menus, and triggering animations in real-world training conditions. If I noticed a rendering stutter, tool lag, or a missed interaction, I could trace it back to a visible performance spike or resource bottleneck right there in the live headset display.

Issue Tracing and Iteration

By combining OVR Metrics Tool feedback with Unity's in-editor logs and profiler, I could pinpoint the source of issues more efficiently. For example, I used it to catch UI stutters caused by overlapping canvas updates and to track down excessive draw calls when multiple evidence items spawned at once. These insights directly informed performance improvements later in development, including batching tweaks and removing effects that were not essential. That ongoing process kept the simulation running smoothly and made sure the experience stayed responsive on the Quest 3.

User Testing

Test Environment

- **Hardware:** Meta Quest 3
- **Software:** Crime Scene Investigation Beta 1.0

Test Objectives & Scope

For this section, I will be having testers experience the CSI mixed reality tool, to put it through its paces using the below test sheet. They'll complete tasks to use all features, scoring each one against set evaluation criteria while also leaving open comments on what worked well and what needs improvement. Once the test rounds are done, I'll pull all the data together and look for patterns, recurring bugs, awkward user flows, or performance hiccups. This will help me prioritise fixes and target upgrades that actually matter to end users. It's how I'll make sure the project is solid before submission and that it meets the real training needs of Humberside Police. The test cases are stored below in the appendix.

(Qualitest, 2023)

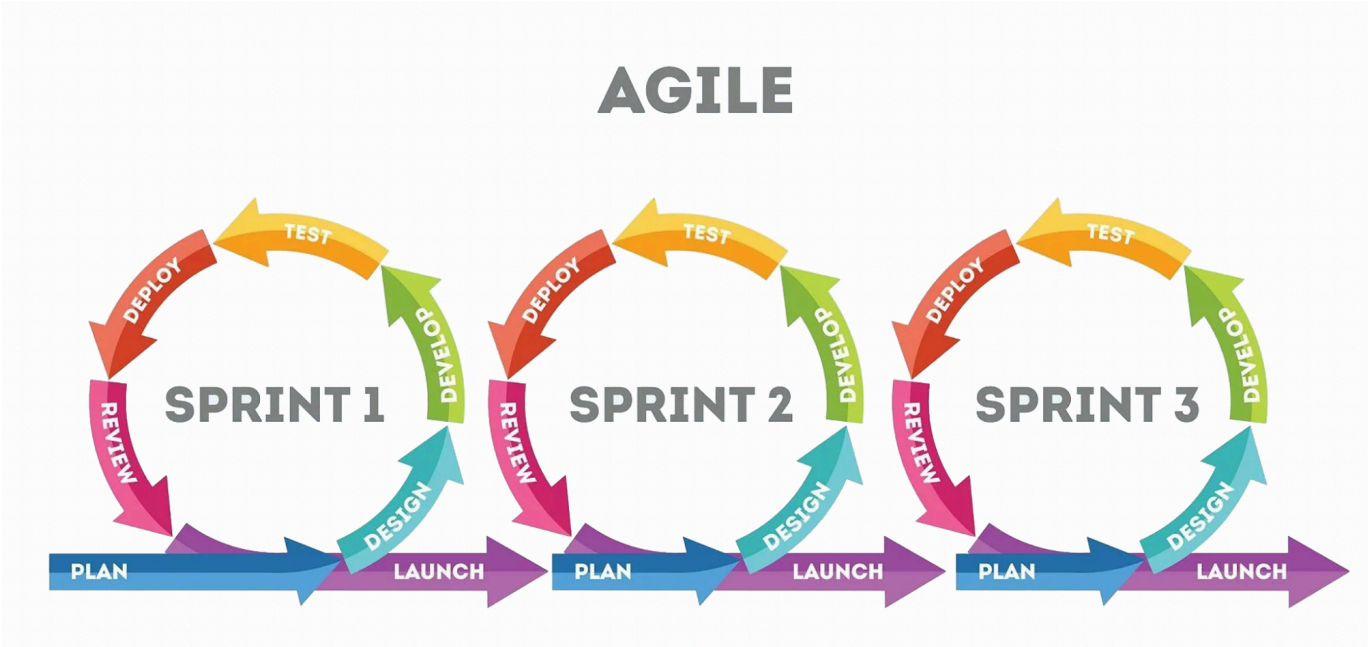
Process and Management

Setting the Plan

From the moment I committed to the "Mixed Reality Game for Training Police Officers" brief, I knew the only way I would hit the standard I was aiming for was by sticking to a clear plan. In week one, I mapped out all the main milestones: getting the concept signed off by Dr Warren Viant and Dr Xinhui Ma, designing the core mechanics, prototyping the investigation tools, building the interface for evidence tracking, and polishing everything for the final demo. I logged it all on a colour coded Gantt chart, week by week, with arrows showing dependencies. This gave me an overview at a glance, like spotting that I could start building the tool spawner interface while benchmarking the Quest 3 passthrough, which saved me almost a week. I will attach a copy of my Gantt Chart in the Appendix.

Working in Sprints

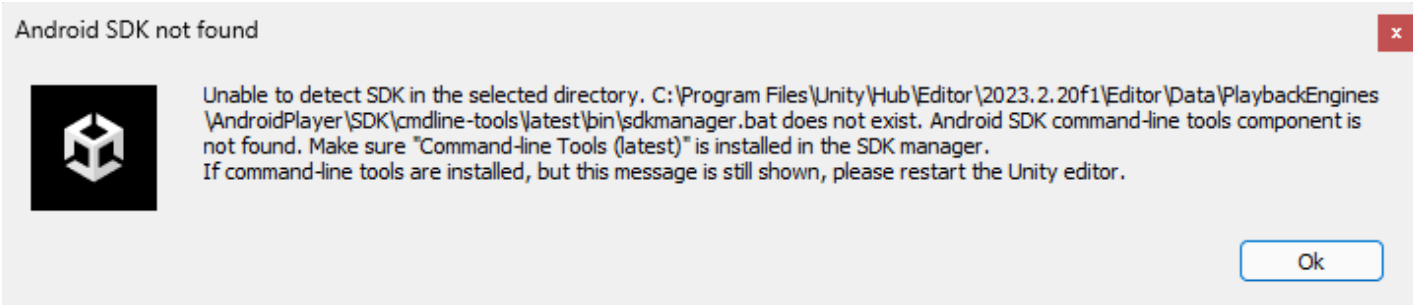
Right from the start, I used an Agile structure. Rather than locking everything down upfront, I broke the work into two week sprints. The first sprint was all about getting the bones of the app together: scene loading and clearing, basic Play and Pause control, and the initial randomisation logic for the three core locations (the bar, the office, and the home). Each morning, I ran a ten minute check-in with myself to review what I had done, list what was next, and flag anything blocking progress. By sprint two, I had the teleport beam working, along with the system that makes held items follow the player, which I showed during a catch-up with Dr Ma.



(SolDevelo, n.d.)

Early Blockers and Fixes

The biggest blocker came early. Getting the app to run on the Quest 3 was much harder than expected. It worked fine in Unity, but every APK crashed. I wasted weeks trying to debug it solo before Jack, another XR student, jumped in and helped me figure it out. It turned out my Unity version did not work well with the Quest 3 runtime. We rolled back to a long term support release, tweaked the SDK settings, and just like that, the build ran smoothly. That fix unlocked sprint three and let me move forward properly.



(Above screenshot taken 10/12/2024)

Avoiding Over Scoping

I almost made the mistake of trying to do too much. At the start, I thought I could cover two or three training areas like evidence collection, weapons safety, and domestic incidents. But Dr Viant pointed out something important. Going deep would earn more marks than spreading myself thin. After thinking it through, I mapped out the pros and cons, and looked at how much time I had left. I kept my focus on the crime scene, and moved everything else to a post demo wishlist. That decision saved the project from drifting and helped me keep the quality high where it mattered most.

Building Key Features

As the project developed, I kept weaving sprint goals into specific features. The tool spawner was a big one. It is a floating menu that follows the player's hand and spawns tools like the scanner, UV torch, brush, and camera. Each tool had its own logic, and I wrote in auto despawn timers and haptic feedback. For the dust brush, I refined the fingerprint detection so it revealed prints progressively, with haptics responding to pressure and direction.

Meetings and Communication

Those weekly Teams meetings with Dr Ma were absolutely essential. They weren't just checkpoints, they were the engine room of the whole project. Each session gave me the space to talk about what I'd built, take on feedback in real time, and tweak the plan. That rhythm kept the pace steady and the direction clear.

Those calls also helped build trust. I could explain my decisions, Dr Ma could steer the direction, and together we kept everything aligned. I think it stopped me drifting into separate tracks and kept the project grounded in real priorities. Every week felt like progress, not just updates.

Managing Time and Staying Organised

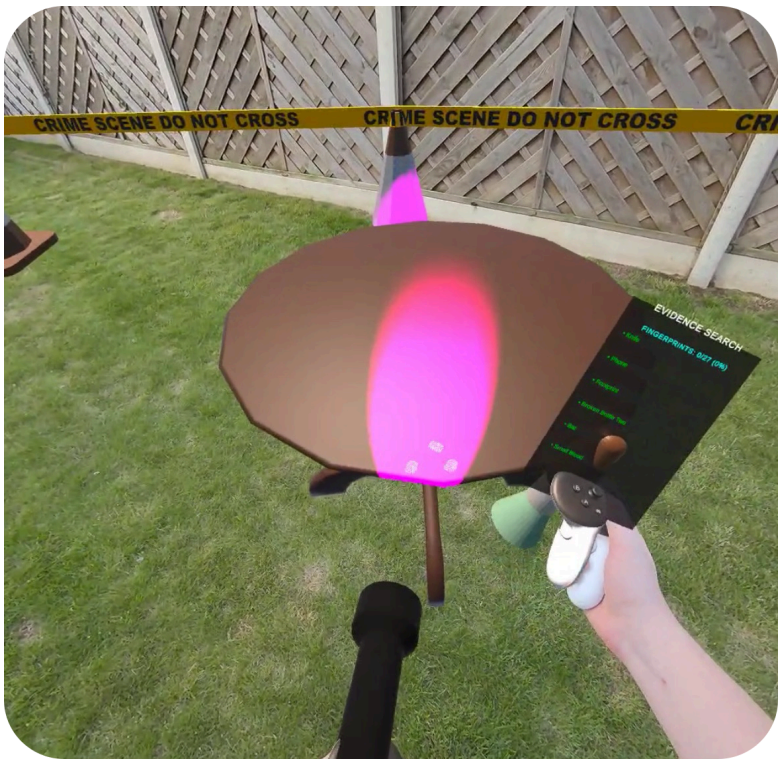
Using a Gantt chart to plan a project made everything more manageable. I started by listing out every task; research, design, testing, implementation, and gave each one a start date, an end point, and a rough estimate of how long it would take. Then I dropped it all onto a calendar grid, drawing bars to show how long each piece would run for. Adding arrows between tasks helped me quickly see what depended on what, which meant if one thing slipped, I knew straight away what else needed to move. I used colours to separate the high priority tasks from the flexible ones, so each week it was obvious where to put my time. Updating the chart regularly also kept the plan realistic. Whether I was working on the interface, fixing movement bugs, or doing final polish, the chart made sure I always knew what was next and when it had to be done.

Final Integration and Demo

By sprint five, the app had come together. It included real time evidence tracking, a mystery toggle that hides evidence names and animated checklist updates. The scene generator supported manual or randomised scenarios, and I added toggles to control evidence count and furniture visibility. Each setting was based on feedback from meetings.

I kept project work and everything else in balance by using a simple priority system. Anything that was both urgent and important went straight to the top of my to-do list. Everything else got pushed to the weekend. I also knew I wasn't going to be at my best every single day, so I paid attention to when I felt most focused. On those days when my head was clear and everything clicked, I made the most of it, even if that meant working late. That way, I got the heavy stuff done when I was sharp, and didn't waste time trying to force it when I wasn't. It helped me stay productive without burning out.

In the final fortnight, I added the visual and audio polish including teleport beams, brush animations, flash burst, shutter sounds, and directional audio. I aimed for sixty frames per second using debug menus and validation tools to monitor performance. Sticking to my sprints, managing risks, and keeping scope tight helped me deliver a polished and technically solid mixed reality training tool on time with all the core features working as intended.

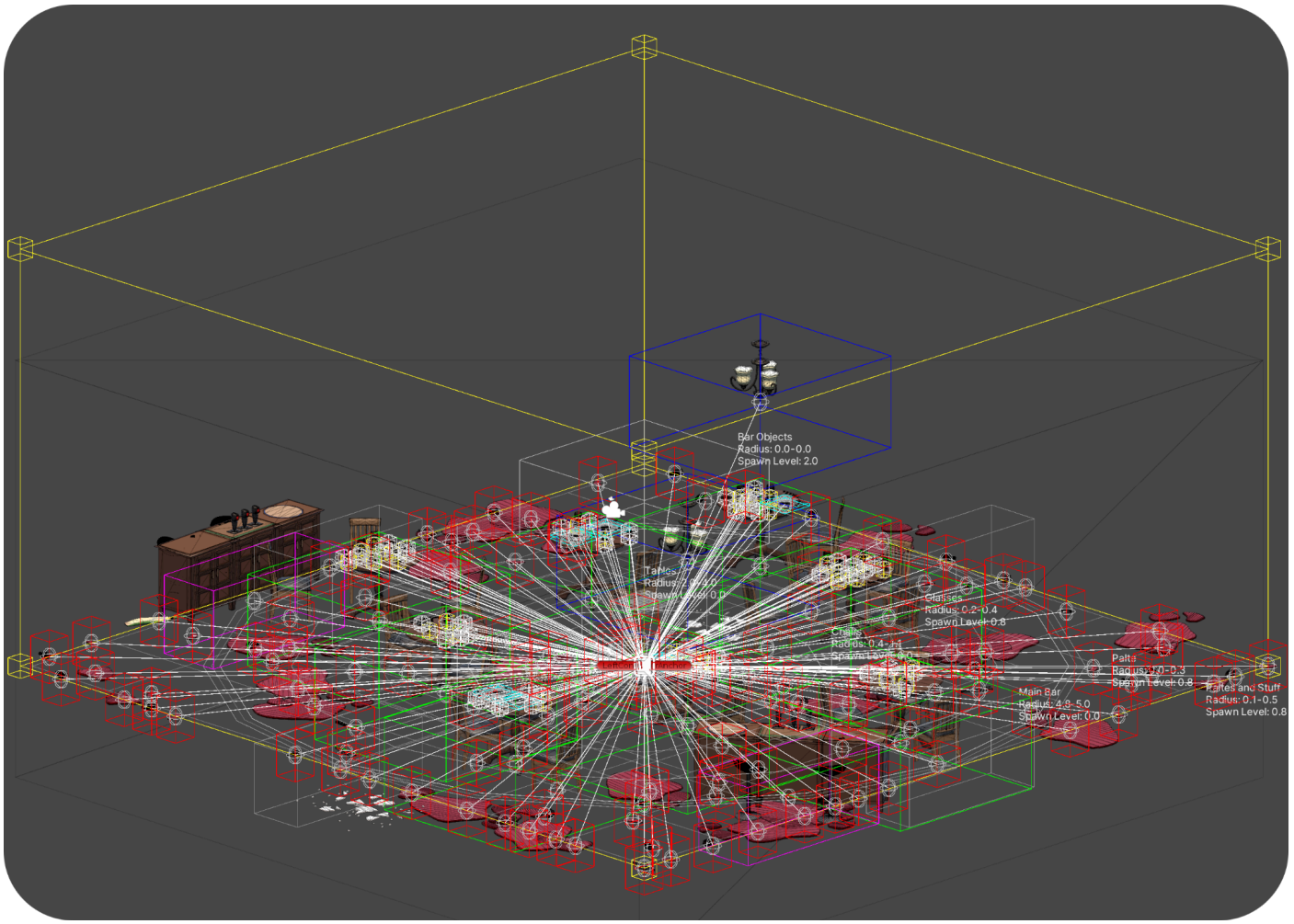


Version 1.1 - Procedural Generation

Introduction and Motivation

Version 1.0 introduced randomised scene layouts that gave each session a sense of freshness and unpredictability, but this often came at the cost of realism. While the visual variety kept things engaging, many of the generated environments lacked any real structure or forensic logic. As objects and items had a pre-determined place they could spawn, it meant that yes, every scene would be new and randomly generated, but only to a certain extent.

Version 1.1 addresses that directly. It brings spatial reasoning and contextual placement into the scene generator by using orbit-based spawn logic, hierarchical object relationships, and real-time configuration control. Now, objects are arranged in ways that reflect how rooms are actually used, chairs arc around tables, glassware clusters near bar counters, and evidence sits where you might expect it after a real incident. The aim was not just to make things different every time, but to make them make sense and to generate scenes that feel like something happened there and invite investigation, not guesswork.



DexterAI (DexterAI.cs)

At the centre of the Version 1.1 generation system sits DexterAI.cs, a custom-built orchestration script that ties the entire procedural framework together. In earlier builds, scene generation was managed through scattered direct calls and isolated logic. That approach worked in simple cases, but it quickly became fragile as new features like revolution logic, hierarchical placement, and live editing were added. DexterAI solves that problem by acting as the brain of the system, a central coordinator that governs how all the subsystems work together without creating tight couplings or race conditions.

Unlike traditional managers that handle everything themselves, DexterAI doesn't spawn objects directly. Instead, it controls the flow of generation by:

- Instantiating and configuring all relevant subsystems at runtime
- Routing lifecycle events like GenerateScene() and ClearScene() to the appropriate modules
- Storing a full snapshot of the current configuration and watching for changes
- Triggering regeneration asynchronously when updates are detected
- Debouncing those updates to prevent constant re-instancing during parameter tuning
- Acting as a shared interface for both visual debugging tools and the placement logic beneath them

This design makes DexterAI the system's single point of truth. Any update, whether it comes from a UI toggle, an inspector change, or an external preset load, flows through DexterAI first. That makes the whole system predictable, modular, and easier to test or extend in future versions.

Connected Modules

DexterAI connects to six custom subsystems:

Module	Responsibility
SceneGeneratorCategories	Determines which object types are enabled and spawns categories accordingly
SceneGeneratorData	Stores the configuration objects (ObjectTypeConfig, RevolutionConfig)
SceneGeneratorDespawn	Handles cleanup of previous scenes in a safe and reusable way
SceneGeneratorHierarchy	Implements parent–child spawn logic (e.g., Tables → Glasses)
SceneGeneratorSpawning	Executes orbit-based placement for objects using revolution maths
SceneGeneratorVisualization	Draws gizmos and debug overlays for orbit paths and layout states

These are all initialised at runtime through:

```
// - INITIALIZATION
private void InitializeComponents()
{
  spawningSystem = new SceneGeneratorSpawning(this);
  hierarchySystem = new SceneGeneratorHierarchy(this);
  despawnSystem = new SceneGeneratorDespawn(this);
  categorySystem = new SceneGeneratorCategories(this);

  if (visualizationSystem == null)
  {
    visualizationSystem = new SceneGeneratorVisualization(this);
  }
  else
  {
    visualizationSystem.SetGenerator(this);
  }

  spawningSystem.SetDependencies(hierarchySystem, despawnSystem);
  visualizationSystem.SetDependencies(spawningSystem, hierarchySystem);
}
```

DexterAI injects itself into each module as a shared reference, giving every system controlled access to global values without introducing tight dependencies. This means that subsystems like revolution logic, spawn point filtering, or evidence placement, can query the current random seed, retrieve central object positions, or read active configuration settings without needing to know how those values are stored or managed. Instead of pulling data from scattered static classes or relying on direct inspector links, everything flows through DexterAI. This keeps the architecture clean, scalable, and easy to debug, especially when tuning parameters live or loading scene presets mid-session.

Modular Generation Cycle

When GenerateScene() is triggered, whether at startup, by loading a preset, or in response to a live configuration change, DexterAI runs through the entire generation sequence in a clearly defined order. It begins by clearing the current scene, resetting all object pools, and zeroing any spatial caches. It then hands off control to each procedural subsystem in turn: environment activation, revolution zone calculation, object categorisation, evidence placement, and finally, visualisation. Each step is timed and tracked, with regeneration debounced to prevent unnecessary reruns during rapid edits.

```
// - SCENE GENERATION
public void GenerateScene()
{
    if (randomizer == null)
        InitializeRandomizer();

    lastGenerationFrame = Time.frameCount;

    if (clearPreviousScene)
        ClearScene();

    allSpawnedObjects.Clear();

    SetupCentralObject();
    ValidateEvidenceQuantities();

    despawnSystem?.ManageExistingObjects();
    spawningSystem?.GenerateObjects();

    CreateConfigurationSnapshot();
    StartCoroutine(NotifySystemsAfterGeneration());
}
```

Configuration Snapshot and Live Monitoring

DexterAI tracks changes through an internal snapshot system that serialises key values from the ObjectTypeConfig and RevolutionConfig files. Each frame, it compares the current configuration against the previous one, checking for differences in spawn radii, object counts, revolution parameters, and even the position or rotation of the central spawn target. If any difference is detected, it flags a regeneration cycle, which is triggered after a short delay using a debounce timer. This prevents the system from overreacting to rapid changes and ensures that generation only happens once the edits settle.

```
// - REAL-TIME CONFIGURATION MONITORING
  private void CheckForConfigurationChanges()
  {
    if (Time.time - lastUpdateCheckTime < updateCheckInterval)
      return;

    lastUpdateCheckTime = Time.time;

    bool hasChanges = false;

    // Check object type configuration changes
    var configChanges = HasObjectTypeConfigurationsChangedDetailed();
    if (configChanges.hasChanges)
    {
      hasChanges = true;
    }

    // Check central position changes
    Vector3 currentCentralPos = GetCentralPosition();
    if (Vector3.Distance(currentCentralPos, lastCentralPosition) > 0.01f)
    {
      lastCentralPosition = currentCentralPos;
      hasChanges = true;
    }

    // Check environment bounds changes
    if (lastEnvironmentBoundsMin != environmentBoundsMin || lastEnvironmentBoundsMax != environmentBoundsMax)
    {
      lastEnvironmentBoundsMin = environmentBoundsMin;
      lastEnvironmentBoundsMax = environmentBoundsMax;
      hasChanges = true;
    }

    // Check seed changes
    if (lastUseCustomSeed != useCustomSeed || (useCustomSeed && lastCustomSeed != customSeed))
    {
      lastUseCustomSeed = useCustomSeed;
      lastCustomSeed = customSeed;
      hasChanges = true;
    }

    if (hasChanges)
    {
      TriggerDebouncedUpdate();
    }
  }
```

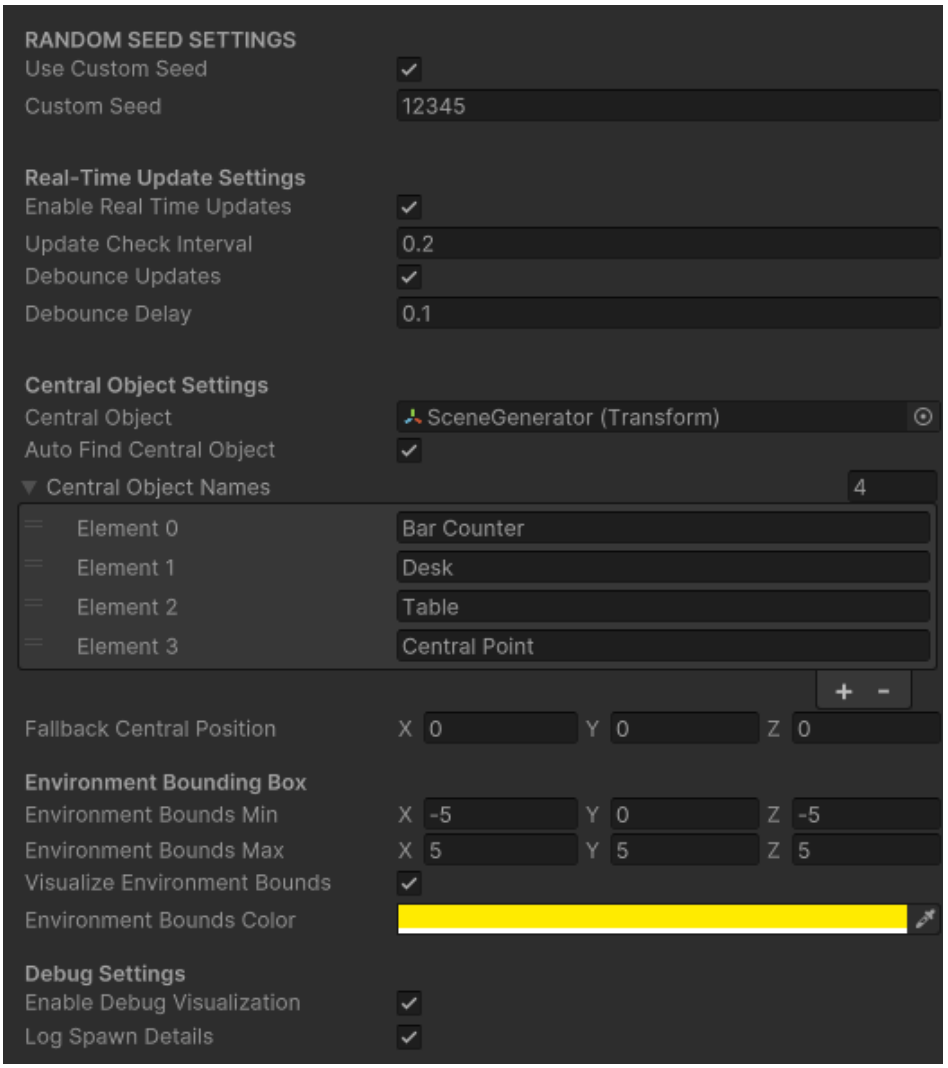
Real-Time Inspector Support

Another important function of DexterAI was supporting development workflows during runtime. Since Unity discards any inspector changes made during play mode, I originally built a custom right-click context system that could save the current generator state, including spawn counts, revolution parameters, and category filters as a JSON file. This made it easy to test and adjust settings live, then commit those changes for future sessions without manually copying values. However, the JSON save functionality was removed before submission, as I didn't have time to finalise it and ensure it was fully functional.

DexterAI exposes public methods like ForceRealTimeUpdate() and SetNewRandomSeed() to trigger updates or reseed randomisation without interrupting the simulation. These utilities make it easy to test specific layouts, replay scenarios, or reset the generator for bug testing.

Communication between systems follows a centralised but decoupled model. Rather than accessing each other directly, all major subsystems communicate through DexterAI. If the spawning system needs to check which categories are enabled, it asks DexterAI, which retrieves the data from SceneGeneratorCategories. If the hierarchy module needs a list of parent objects for table-child relationships, DexterAI provides the current list of spawned items. If the visualiser needs to draw orbit arcs or spawn regions, it queries DexterAI for the relevant config values and spawn metadata.

This central routing layer ensures each module remains isolated and testable on its own, while still participating in a unified scene-building pipeline. The result is a powerful, maintainable architecture that supports complexity without introducing tight coupling or runtime instability.



Why It Matters

Without DexterAI, the scene generation system would either spiral into a monolithic structure, packed with tightly bound method calls scattered across dozens of classes, or collapse into fragmentation, where no module has awareness of the broader generation context. DexterAI solves both extremes by acting as a smart, reactive orchestrator that keeps the system modular, clean, and extensible. Every subsystem knows its role, but none of them need to know how the others work.

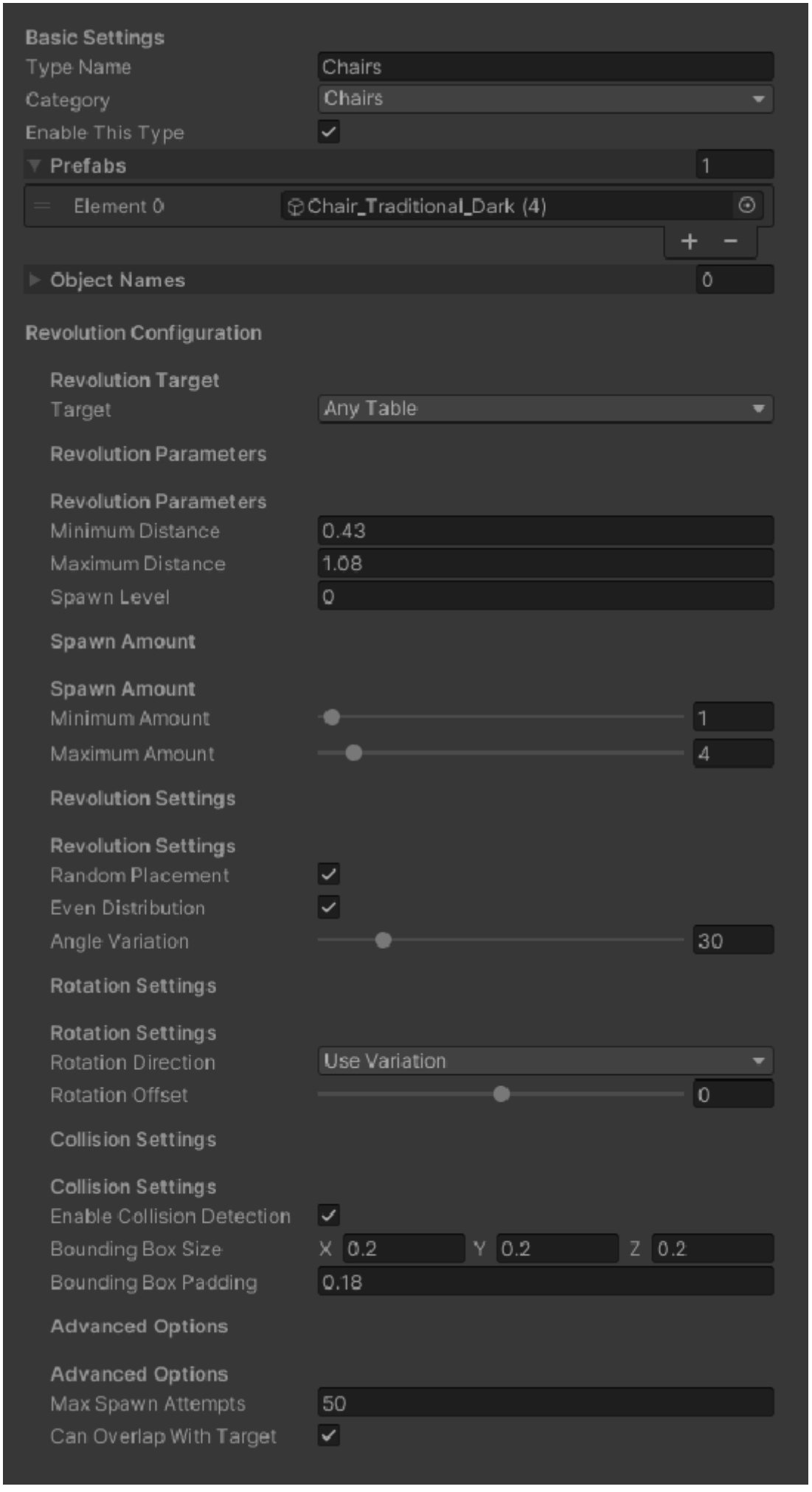
That modularity is what makes future upgrades possible. Whether it's adaptive difficulty that reacts to trainee skill level, AI-driven story progression, or branching narrative events based on player actions, each new feature can plug into DexterAI's central flow without rewriting the foundation.

Revolution-Based Spawning

The most significant technical upgrade in Version 1.1 is the shift to revolution-based spawning. Rather than relying on random scatter or hardcoded spawn points, objects are now placed in circular or orbital patterns around dynamic targets, like chairs naturally arranged around tables or bottles distributed along bar counters. These targets aren't specified by name, but are resolved in real time using object categories and their relationships.

At the centre of this system is the new RevolutionConfig, which sits inside each ObjectTypeConfig. This controls how objects behave when orbiting a target. For example, chairs might define a minimum and maximum distance from a table, a requirement to face inwards, angular spacing variation to break symmetry, a cap on how many can be placed, and a rule that prevents overlapping with other objects. This setup allows for natural-looking arrangements without the repetition or randomness of earlier builds.

Every object spawned this way is checked against spatial constraints using bounding boxes and placement logic. If a position would cause an intersection or visual issue, it's skipped or adjusted, unless that category has been explicitly allowed to intersect under certain conditions. The result is a cleaner, more intentional scene layout where everything appears positioned by design, not by accident.

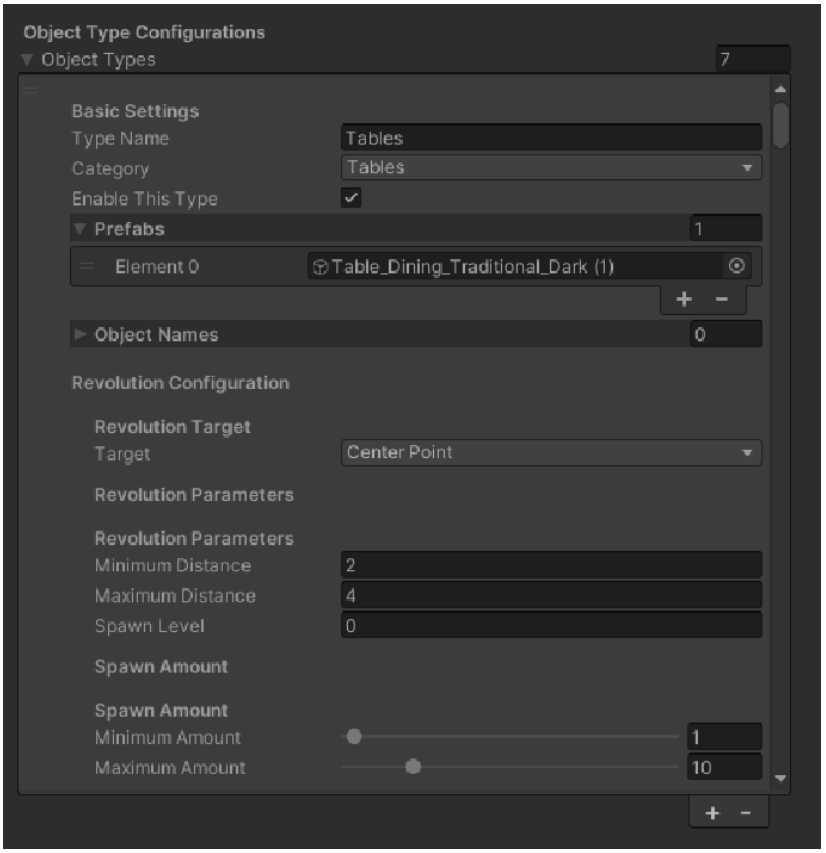

Basic Settings
Type Name
Chairs
Category
Chairs
Enable This Type
Prefabs
1
Element 0
Chair_Traditional_Dark (4)
Object Names
0
Revolution Configuration
Revolution Target
Target
Any Table
Revolution Parameters
Revolution Parameters
Minimum Distance
0.43
Maximum Distance
1.08
Spawn Level
0
Spawn Amount
Spawn Amount
Minimum Amount
1
Maximum Amount
4
Revolution Settings
Revolution Settings
Random Placement
Even Distribution
Angle Variation
30
Rotation Settings
Rotation Settings
Rotation Direction
Use Variation
Rotation Offset
0
Collision Settings
Collision Settings
Enable Collision Detection
Bounding Box Size
X 0.2
Y 0.2
Z 0.2
Bounding Box Padding
0.18
Advanced Options
Advanced Options
Max Spawn Attempts
50
Can Overlap With Target

Hierarchical Object Logic

Scene generation now follows a structured two-pass process. In the first pass, all major objects such as tables, counters, or large furniture, are placed. These form the structural foundation of the environment. In the second pass, the SceneGeneratorHierarchy system steps in to populate these parent objects with smaller, context-specific items like glasses on tables, bottles behind bars, or decorations on shelving.

Each parent-child link is governed by its own ruleset. These rules define how many children can spawn, the distance from the parent, the angle spread, and whether the alignment must be strict (like cups facing forward) or relaxed (like scattered bottles). These rules bring cohesion to the scene and ensure that the placement of objects respects both function and believability. Tables spawn items that would realistically be on tables, shelves hold items that make sense at that height, and nothing feels out of place.

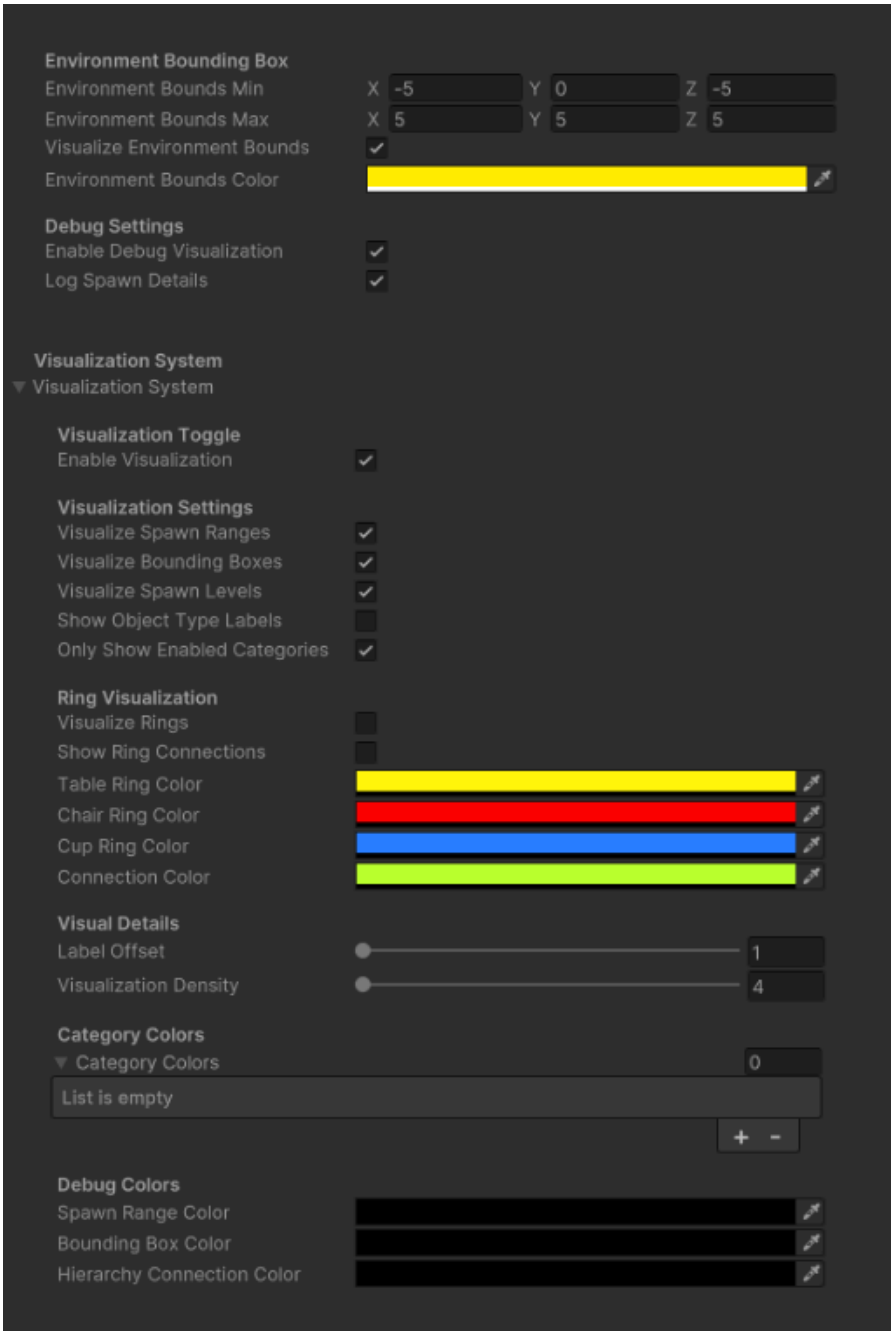
DexterAI handles the coordination. Once the first pass is done and all major objects are registered, it triggers the hierarchy phase, feeding each parent object its category-specific spawn rules and distributing available slots in a balanced way. The outcome is a scene that looks like it was set up with purpose, not just randomly filled.



Visualisation and Debug Tools

To support layout tuning during development, I added a full visualisation layer using Unity Gizmos. Each spawn orbit is rendered live in the editor with colour-coded rings, radius arcs, and labels showing which object is which. Bounding boxes are drawn around all spawned objects, making it easy to spot overlap problems at a glance. Parent-child relationships are shown with connecting lines, giving a clear view of hierarchy chains and placement logic.

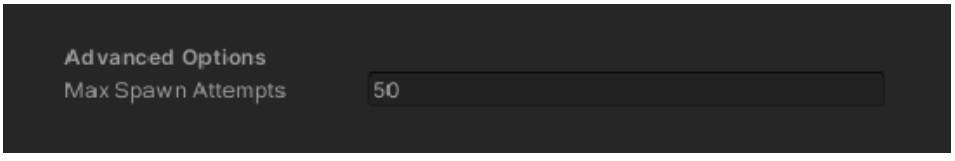
The visualisation system itself is managed by the SceneGeneratorVisualization module, but all configuration and triggering are handled by DexterAI. This keeps the visual output tightly synced with whatever rules and parameters are currently active. It was especially helpful during debugging, letting me pinpoint issues like misaligned seating or overpacked bar counters without needing to step through code or watch runtime output.



Performance and Modularity

Even with all its added complexity, the generator remains quick and efficient. Spawn attempts are capped to avoid long loops, collision checks are lightweight, and every system runs independently under DexterAI's coordination. Scenes with hundreds of objects can still regenerate in under a second. If an object fails to place. For example, due to overlap or no available space, it's quietly skipped without breaking the layout.

Because each system is modular and centrally managed, I can add new rules, object types, or behaviours without risking existing functionality. The result is a procedural engine that doesn't just produce variety for its own sake, it produces scenes that make sense, with structure, balance, and a clear logic behind them.



Evaluation and discussion of results

Evaluation of Project Outcomes

Achievement of Project Goals

At the start of this project, my main goal was to explore whether a serious mixed reality game could genuinely improve the way new forensic investigators search for and collect evidence. I wanted to go beyond static classroom learning and create something immersive and practical; an experience that reflected the real challenges of crime scene work. I believe the final product, delivered as a standalone Meta Quest 3 application, achieves that aim.

The system supports a variety of environments, uses procedural content generation for randomisation, and offers realistic evidence handling alongside real-time feedback through the checklist and scoring interface. These features were not added for show, they were chosen carefully to support learning. They help reinforce good habits, build confidence through repetition, and introduce new challenges at a pace the trainee can handle.

As development progressed into Version 1.1, I refined the procedural system further using a new modular controller called DexterAI, which allowed curated scene creation with precise control over item types, placement, and spawn logic. The move from randomised generation to rule-based, configurable scenes made it easier to target specific learning outcomes, especially in structured training sessions or assessments.

Even though feedback is still early, the signs are encouraging. Testers said the scenes felt natural, and that uncovering fingerprints with the dust brush added a real sense of discovery. The checklist helped them stay focused and gave immediate feedback on what they had missed, prompting them to go back and try again. That replay value is exactly what I was aiming for. It encourages procedural learning and helps solidify best practices through repeated use.

More than anything, the tool makes forensic training feel more hands-on and rewarding rather than slow or repetitive, and for a serious learning application, that is a real success.

Reflection on Challenges

Technical Challenges

From a technical point of view, the biggest challenge was making everything work smoothly on the Meta Quest 3. Early on, I ran into a frustrating issue where builds worked perfectly in Unity's editor but crashed the moment they were deployed to the headset. After a lot of digging and having burned an awful lot of time, I traced the problem to an incompatibility between the latest version of Unity and the Quest 3 runtime. The solution involved rolling back to Unity 2022 Long Term Support and manually aligning all plugin versions. That experience taught me a valuable lesson: when working with new hardware, it is safer to pick a stable and well-supported build setup from the beginning, even if it means sacrificing the latest features.

Getting the event-driven architecture stable was more work than I expected. Unity's default lifecycle can be chaotic when multiple systems try to act at the same time. For example, the checklist might query the scene before it finishes generating. I fixed this by adding a simple event bus and spacing out system updates with coroutines. That way, each system waited for the right signal before doing anything, resulting in a cleaner flow and far fewer timing bugs.

In Version 1.1, these architectural changes paid off. DexterAI required far more communication between systems, such as the evidence registry, scene spawning logic, and the fingerprint detector. To avoid race conditions and redundant calls, I expanded the event bus and added condition checks for every major lifecycle event. Another technical improvement in 1.1 was enabling scene tuning live in the Unity inspector, so trainers or developers could tweak category weights, spawn densities, and placement radii without rebuilding. I had planned to let users export these changes as JSON files directly from play mode, but that feature was removed from the final build due to time constraints, I simply didn't have time to make it fully functional.

Ethical and Legal Considerations

Early in the design phase, I identified several ethical and legal risks, particularly around emotional distress and data privacy. The application does not record any personally identifiable information, which makes complying with GDPR effortless. On the emotional side, I remained aware that simulated crime scenes, even in a stylised form, can still provoke discomfort. Although my current test scenarios avoid graphic or disturbing content, I designed the system to be scalable. In a commercial or academic deployment, each session could begin with a consent form and an optional mental health check-in. The system could also monitor indirect distress signals, such as sudden exits, reduced interaction, or physical hesitation, and flag these patterns for instructor review. From there, it would be easy to layer in custom filters or adaptive content settings based on the trainee's profile and preferences.

The updated procedural system added an unexpected benefit here. Because scenarios are now defined using configurable parameters and tag-based object types, it's far easier to filter out sensitive themes or tailor scene intensity to different user profiles. Whether it's avoiding certain room types or excluding high-risk content categories, this makes the system more flexible and ethically resilient for future deployments.

Future Improvements and Developments

Potential Features

Now that the core system is stable and tested, several clear improvements stand out as the next steps. The first is expanding scene complexity. At the moment, the tool supports three base environments: a bar, an office, and a home, but the generator is fully modular. It could easily be extended to include outdoor locations, vehicles, or public spaces. This would increase scenario diversity and provide more complex challenges for advanced trainees.

Another obvious evolution is narrative structure. The current system generates realistic mixtures of evidence, but the links between items are superficial. I want to move towards a narrative planner that ties items together around motive, timeline, or character involvement, turning the exercise into a proper investigation. This would also open the door to multi-phase cases, where evidence appears conditionally based on user actions.

Much of this groundwork has already been laid by DexterAI, which includes object categorisation and modular spawn logic. By attaching metadata to items such as "related to motive" or "scene trigger", I can build a branching logic system that governs not just what appears, but when and why. It's an achievable next step, and one that would add major depth to the training experience.

Looking further ahead, I'm interested in voice-based tutoring. A virtual instructor could offer gentle guidance if the user stalls or overlooks key evidence, providing the kind of soft mentorship that's hard to simulate in solo VR sessions. Natural language understanding could also let users ask questions like "What should I do next?" or "Does this look suspicious?", encouraging reflection and deeper thinking. Combined with dynamic scene adaptation, this could make each session feel tailored and intelligent.

Lastly, I see strong potential in accessibility enhancements. Support for subtitles, contrast themes, and more robust hand tracking would extend usability across different user groups. These features are relatively simple to implement but would increase the platform's value to institutions aiming to support diverse needs.

Market Expansion

Although this project was designed for forensic training, the underlying framework could be adapted for many other fields. With only minor changes, the tool could support emergency response training, such as triage in disaster zones, or help simulate hospital intake and post-mortem reporting for medical education.

In the law enforcement world, the tool could be positioned as a cost-effective supplement to traditional training. It offers a repeatable, portable environment where new recruits can build practical skills before being sent to real scenes. For universities, it would fit naturally into forensic science courses, giving students a safe and self-guided way to apply what they learn in lectures. There is even potential in the public education space, perhaps as a simplified forensic escape room for schools or museums, designed to engage the public while teaching basic investigative principles.

The refined procedural system is a key enabler here. With minimal effort, scenes can be reskinned, logic tweaked, and evidence types redefined to suit a completely different training goal. Whether it's airport security drills or environmental site inspections, the platform's adaptability makes it a viable candidate for commercial licensing or white-label deployment. Even as a Unity toolkit for developers, the system is modular enough to serve as the backbone for a wide range of spatial training applications.

Because it runs entirely on the Meta Quest 3, without needing external hardware, it's easy to roll out at scale, even for organisations with limited technical infrastructure. That accessibility, combined with a robust underlying architecture and growing flexibility, makes the case for further investment very strong.

Final Conclusions

Project Management

Looking back at my original plan, I'm pleased with how closely I stuck to the Gantt chart and sprint framework, despite the initial SDK technical hiatus and the necessity of a time extension. An Agile methodology was the right choice as it let me build in short cycles, test and fail early, and avoid getting bogged down in any one system. The weekly meetings with my project supervisor kept the project focused, avoided scope-creep, and the dependency mapping in my Gantt chart meant I always knew which task to prioritise next. Even when things went wrong, or very very wrong like the Quest 3 SDK bug, I had just enough breathing room to adapt without slipping off schedule.

Overall Conclusion

In the end, this project wasn't just about building a working mixed reality app. It was about testing a deeper idea: that well-designed technology can bridge the gap between theoretical learning and practical application, especially in high-stakes, detail-sensitive fields like forensic investigation. The core hypothesis was that even when training alone without a live instructor or physical crime scene, a trainee could still develop sharp procedural instincts and real investigative confidence if given the right tools and structured feedback.

What I've seen throughout this project supports that idea. The fluidity of the interactions, from the intuitive evidence detection systems to the step-by-step fingerprint revelation process, suggests that the tool doesn't just simulate investigation, it teaches it. Testers described how easy it was to learn, how immersive it felt, and how satisfying it was to complete an investigation with real steps and visual confirmation. That feedback loop, observe, act, receive response, is what embeds skills. The fact that it all happens inside a standalone headset makes it portable, repeatable, and accessible.

From a technical perspective, I've built something that's scalable and solid. The core architecture is modular enough to support new environments or deeper logic without needing to be restructured. Its performance on the Quest 3 shows that serious training doesn't have to rely on expensive or tethered VR setups. On top of this, it can be easily converted into a VR-only solution for older headsets like the Quest 2.

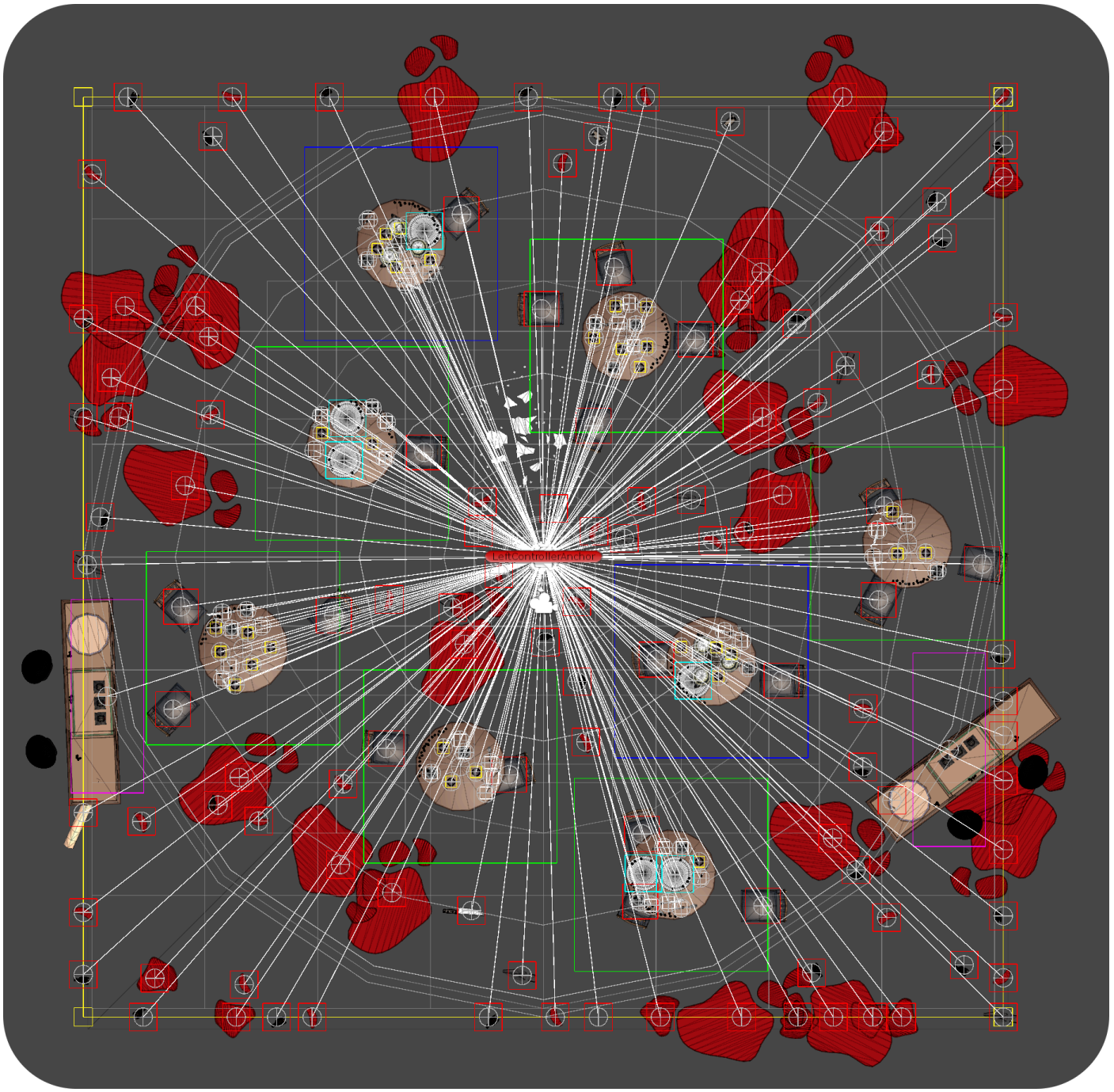
What really sets the system apart, however, is the advancement introduced in Version 1.1. This release redefines scene generation as more than randomised layouts. With the addition of DexterAI, a central orchestration system that coordinates all generation, validation, and real-time tuning, the scenes now evolve with structure and internal logic. Instead of chaotic object scatter, Version 1.1 uses orbit-based spawning, hierarchical placement, and real-time configuration tracking to build crime scenes that feel coherent, believable, and purposeful.

The integration of live parameter editing and configuration snapshots means that scenes can be tuned mid-session, then saved for future use without restarting the application. Developers and instructors can create highly specific layouts or training presets in seconds. This marks a shift from static learning environments to dynamic, instructor-guided simulations. It's a system that adapts to both the learner and the scenario.

There's more to do, of course. Adding narrative complexity, stronger AI-driven logic, emotional safety features, and broader scene diversity are clear next steps. Accessibility could also be improved, especially for trainees with different physical needs or learning styles. But the foundation is not just functional, it's future-ready. It shows that immersive training can be realistic and engaging without losing clarity or educational focus.

With the right partnerships, more feedback, and additional development time, I'm confident this tool could reshape how new investigators are trained. It provides the architecture and interaction design that bridge the gap between classroom theory and real-world practice, between knowing what to do and actually doing it. That's the gap this project set out to close, and with Version 1.1, it closes it even further.

“It may be virtual, but the skills it builds are not.”



References

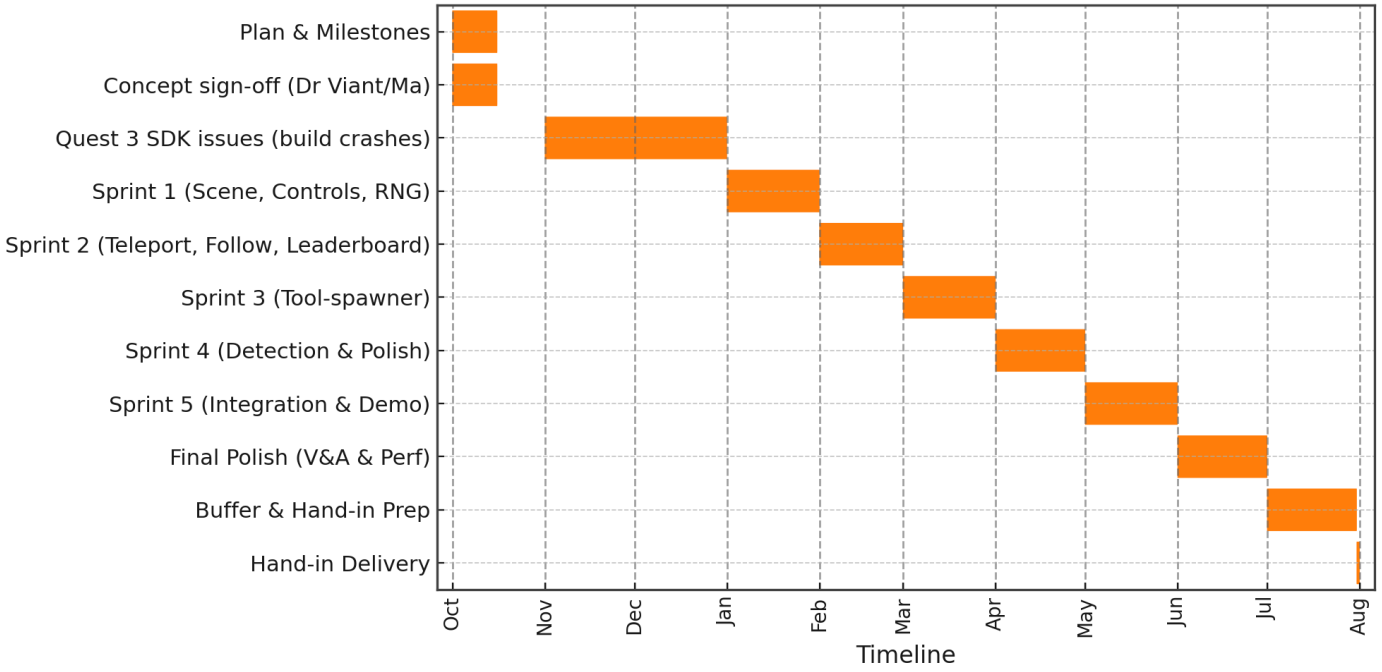
- Apex Officer. (2023). Crime Scene Investigation VR Training Simulator – Apex Officer Demo [video]. YouTube. Available at: https://www.youtube.com/watch?v=Iz7k2THJnLE&ab_channel=ApexOfficer [Accessed 14 Mar. 2025].
- AWE Me. (2023). Building VR Environments in Unity with Realistic Lighting [video]. YouTube. Available at: https://www.youtube.com/watch?v=es9d1y1hYfM [Accessed 22 Nov. 2024].
- College of Policing. (2022). Briefing and Debriefing – Authorised Professional Practice. [online] Available at: https://www.college.police.uk/app/operations/briefing-and-debriefing [Accessed 5 Mar. 2025].
- Code Monkey. (2022). Unity XR Toolkit – Full Setup & Tutorial [video]. YouTube. Available at: https://www.youtube.com/watch?v=-tVfg8Zaqos&t=251s [Accessed 22 Nov. 2024].
- Community GameDev.tv. (2024). Meta Quest 3 Not Connecting – Community Thread. [online] Available at: https://community.gamedev.tv/t/meta-quest-3-not-connecting/242010 [Accessed 16 Jan. 2025].
- Dilmer Valecillos. (2023). Meta Quest Hand Tracking with Unity XR Toolkit – Complete Guide [video]. YouTube. Available at: https://www.youtube.com/watch?v=QHMtVEX9O0Y [Accessed 19 Apr. 2025].
- GodotVR. (2024). XR Template Issue #27 – Meta Quest Compatibility Problems. [online] GitHub. Available at: https://github.com/GodotVR/godot-xr-template/issues/27 [Accessed 17 Nov. 2024].
- Hornby, R. (2024). Meta Quest 3 vs Meta Quest 3S: A VR Headset Head-to-Head. Laptop Mag. [online] Available at: https://www.laptopmag.com/gaming/vr/meta-quest-3-vs-meta-quest-3s-a-vr-headset-head-to-head [Accessed 29 Jul. 2025].
- IEEE. (2024). Augmented Reality Support System for Crime Scene Investigation. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/document/10278406 [Accessed 12 Jul. 2025].
- IEEE. (2024). Mixed Reality in Forensic Education: A Systematic Literature Review. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/document/10329468 [Accessed 5 Jul. 2025].
- IEEE. (2024). AI-Driven Scene Generation for MR Crime Scene Simulations. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/document/10469037 [Accessed 19 Jun. 2025].
- IMDb. (n.d.). Criminal Minds (2005–2020). [online] Available at: https://www.imdb.com/title/tt0452046/ [Accessed 13 Dec. 2024].
- IMDb. (n.d.). CSI: Crime Scene Investigation (2000–2015). [online] Available at: https://www.imdb.com/title/tt0247082/ [Accessed 13 Dec. 2024].
- IMDb. (n.d.). Dexter (2006–2013). [online] Available at: https://www.imdb.com/title/tt0773262/ [Accessed 13 Dec. 2024].
- Lawrence, S. (2025). Honours Stage Project. [online] GitHub. Available at: https://github.com/samuelbslawrence/Honours-Stage-Project/tree/main [Accessed 5 Nov. 2024].
- Mashable. (2023). Meta Quest 3 Reviews Are In – Is It Worth the Upgrade?. [online] Available at: https://mashable.com/article/meta-quest-3-reviews-are-in [Accessed 3 Feb. 2025].
- Meta. (n.d.). OVR Metrics Tool – Meta Quest. [online] Available at: https://www.meta.com/en-gb/experiences/ovr-metrics-tool/2372625889463779/ [Accessed 27 Jan. 2025].
- National Center for Biotechnology Information. (2012). Expertise in crime scene examination. [online] PubMed. Available at: https://pubmed.ncbi.nlm.nih.gov/22768643/ [Accessed 17 May. 2025].
- NeoPangea. (n.d.). Forensic Detective. [online] Available at: https://www.neopangea.com/portfolio/nbc-universal/forensic-detective.html [Accessed 29 Apr. 2025].
- Promega. (2023). Do Forensic Scientists Experience Trauma at Work? Unpacking What We Know About Stress, Trauma, and Burnout in a Forgotten Population. The ISHI Report, February 2023. [online] Available at: https://promega.foleon.com/theishireport/the-ishi-report-february-2023/do-forensic-scientists-experience-trauma-at-work-unpacking-what-we-know-about-stress-trauma-and-burnout-in-a-forgotten-population [Accessed 26 May. 2025].

- Qualitest. (2023). Best Practices for QA Testing in AR/VR/MR Projects. [online] Available at: https://www.qualitestgroup.com/insights/blog/best-practices-qa-testing-ar-vr-mr/ [Accessed 3 Jul. 2025].
- Reid, D. and Moray, N. (2024). Mixed Reality Forensic Training for Police Work: Exploring Procedural and Psychological Engagement. [pdf] University of Glasgow. Available at: https://eprints.gla.ac.uk/337319/2/337319.pdf [Accessed 11 Jul. 2025].
- ResearchGate. (2024). Designing and Evaluation of a Mixed Reality System for Crime Scene Investigation Training: A Hybrid Approach. [online] Available at: https://www.researchgate.net/publication/381703381_Designing_and_evaluation_of_a_mixed_reality_system_for_crime_scene_investigation_training_a_hybrid_approach [Accessed 20 Jun. 2025].
- Road to VR. (2024). Meta CTO Talks Quest 3 Battery, Horizon OS & v76 Update. [online] Available at: https://www.roadtovr.com/meta-cto-quest-3-battery-horizon-os-v76-update/ [Accessed 1 May. 2025].
- ScienceDirect. (2024). Investigating MR Systems for Real-World CSI Training Impact. Computers in Human Behavior, 153, 107478. [online] Available at: https://www.sciencedirect.com/science/article/pii/S1355030624000273 [Accessed 24 Jun. 2025].
- SolDevelo. (n.d.). Is Agile Always the Best Solution for Software Development Projects?. [online] Available at: https://soldevelo.com/blog/is-agile-always-the-best-solution-for-software-development-projects/ [Accessed 6 Mar. 2025].
- Southampton Institutional Repository. (2024). Evaluating the Use of Mixed Reality in CSI Training Through the Integration of the Task-Technology Fit and Technology Acceptance Model. [pdf] Available at: https://eprints.soton.ac.uk/500347/1/Evaluating_the_Use_of_Mixed_Reality_in_CSI_Training_Through_the_Integration_of_the_Task-Technology_Fit_and_Technology_Acceptance_Model.pdf [Accessed 25 Jun. 2025].
- Stack Overflow. (2021). You are missing the recommended SDK/JDK/NDK – Install the recommended version using SDK Manager. [online] Available at: https://stackoverflow.com/questions/69233386/you-are-missing-the-recommended-sdk-jdk-ndk-install-the-recommended-version-usi [Accessed 2 Nov. 2024].
- Technologies. (2024). Mixed Reality Training Systems for Forensic Education. Technologies, 13(8), p.315. [online] MDPI. Available at: https://www.mdpi.com/2227-7080/13/8/315 [Accessed 20 Jun. 2025].
- University of Essex Repository. (2024). A Framework for Crime Scene Training with Immersive Technologies. [pdf] Available at: https://repository.essex.ac.uk/36913/1/Manuscript.pdf [Accessed 28 Jun. 2025].

Appendix

Gantt Chart

I built a straightforward Gantt chart to lay out all the major tasks and milestones for the project. It gave me a clear view of what was happening when, and how each part connects to the next. By marking start and end dates, showing what depends on what, and tracking progress along the way, it keeps everything organised. It gave me the ability to quickly spot what needed doing, what was coming up, and whether I was still on track overall. Below is a condensed version of my Gantt Chart that I used to organise and manage my project.



Test Case 01

Test Case ID: TC-01

Tester Name: Ethan Lutkin

Role: 2nd Year Business Economics Student

Version: Beta 1.0

Experience Level: Intermediate

Criterion	Mark out of 10
UI Design	7/10
Ease of Learning	9/10
Navigation & Workflow	10/10
Speed/Throughput	10/10
Smoothness	9/10
Latency/Responsiveness	8/10
Feature Completeness	7/10
Accuracy	7/10
Reliability	10/10
Satisfaction/Recommendation	8/10
Total	86/100

Strengths	Areas for Improvement
- The tools were seamless, and responsive. - The variety of different scenes was cool and surprising. - How smooth the game ran, and how he didn't run into any crashes or stuttering. - He also liked how he played it in his garden and was impressed at the versatility.	- Prompt to scroll in the menu. - The small boundary was restrictive. - Having boundary showing could be useful, if there was no boundary that would be even better. - The design of menus could be updated/modernised. - Taking pictures of fingerprints was difficult, as he couldn't determine which he had already photographed.

Test Case 02

Test Case ID: TC-02

Tester Name: Spencer Trott

Role: 2nd Year Computer Science Student

Version: Beta 1.0

Experience Level: Advanced

Criterion	Mark out of 10
UI Design	4/10
Ease of Learning	5/10
Navigation & Workflow	7/10
Speed/Throughput	9/10
Smoothness	9/10
Latency/Responsiveness	8/10
Feature Completeness	8/10
Accuracy	8/10
Reliability	10/10
Satisfaction/Recommendation	7/10
Total	75/100

Strengths	Areas for Improvement
- He liked the camera and dust brush. - Impressed with on the fly generation.	- More advanced UI and settings. - Accessibility improvements. - More realistic/photorealism.

Test Case 03

Test Case ID: TC-03

Tester Name: Matthew Lawrence

Role:

Version: Beta 1.0

Experience Level: Intermediate

Criterion	Mark out of 10
UI Design	7/10
Ease of Learning	9/10
Navigation & Workflow	9/10
Speed/Throughput	10/10
Smoothness	9/10
Latency/Responsiveness	9/10
Feature Completeness	7/10
Accuracy	10/10
Reliability	10/10
Satisfaction/Recommendation	8/10
Total	88/100

Strengths	Areas for Improvement
- Randomised evidence scene creation is exciting, as is the prospect of contextual randomisation in future development - Affordable choice of equipment increased accessibility - Modular design given clear opportunities for logical development of features including new scene environments	- I know there are fewer points for gameplay including a finished product, but more of this obviously would help with engagement and utility. - Photo realistic artifact and environmental image creation

Test Case 04

Test Case ID: TC-04

Tester Name: Samantha Lawrence

Role:

Version: Beta 1.0

Experience Level: Beginner

Criterion	Mark out of 10
UI Design	9/10
Ease of Learning	10/10
Navigation & Workflow	9/10
Speed/Throughput	10/10
Smoothness	10/10
Latency/Responsiveness	9/10
Feature Completeness	8/10
Accuracy	10/10
Reliability	10/10
Satisfaction/Recommendation	10/10
Total	75/100

Strengths	Areas for Improvement
- like the ability to grab and hover tools.	- More realistic visuals

Test Case 05

Test Case ID: TC-05

Tester Name: Spencer Trott

Role: 2nd Year Computer Science Student

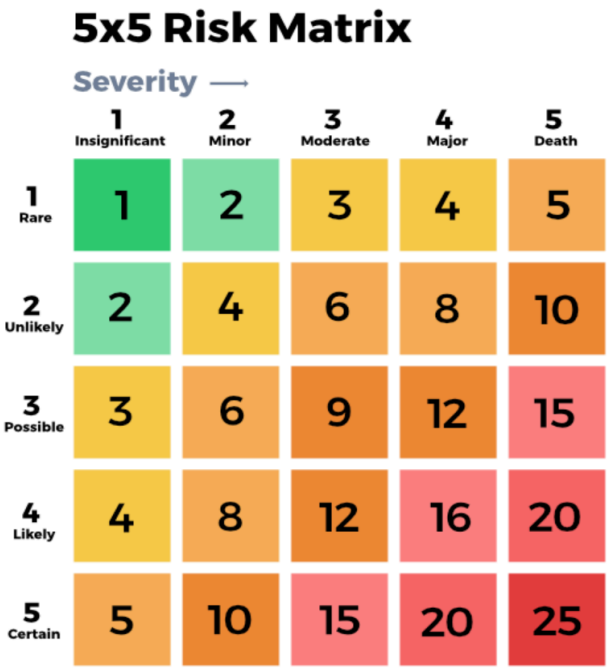
Version: Beta 1.1

Experience Level: Advanced

Criterion	Mark out of 10
UI Design	3/10
Ease of Learning	5/10
Navigation & Workflow	7/10
Speed/Throughput	8/10
Smoothness	9/10
Latency/Responsiveness	9/10
Feature Completeness	10/10
Accuracy	9/10
Reliability	10/10
Satisfaction/Recommendation	10/10
Total	80/100

Strengths	Areas for Improvement
- Like the large scene - Impressed with on the fly generation. - Amount of detail - Evidence Markers - Lots of objects scattered realistically	- More advanced UI and settings. - Accessibility improvements. - More realistic/photorealism. - Add more environments

Risk Matrix



12

Technical Challenges:

Developing my game involves technical elements, such as precise tracking and real-time rendering. I know that bugs and performance issues can and will arise.

6

User Safety:

Since my game encourages players to move around in real-world environments, I need to be mindful of user safety. Accidents can happen if players aren't aware of their surroundings.

4

Privacy Concerns:

Collecting data from users is important. I'll need to ensure compliance with data protection laws and be transparent about how I use their information.

15

Market Competition:

The VR/MR space is competitive, new entrants need to offer something unique. However I have not seen anything that comes close to what I would want to do.

8

User Experience:

Balancing engaging gameplay with MR elements can be challenging. Poor user experience can lead to negative reviews and low retention rates.

9

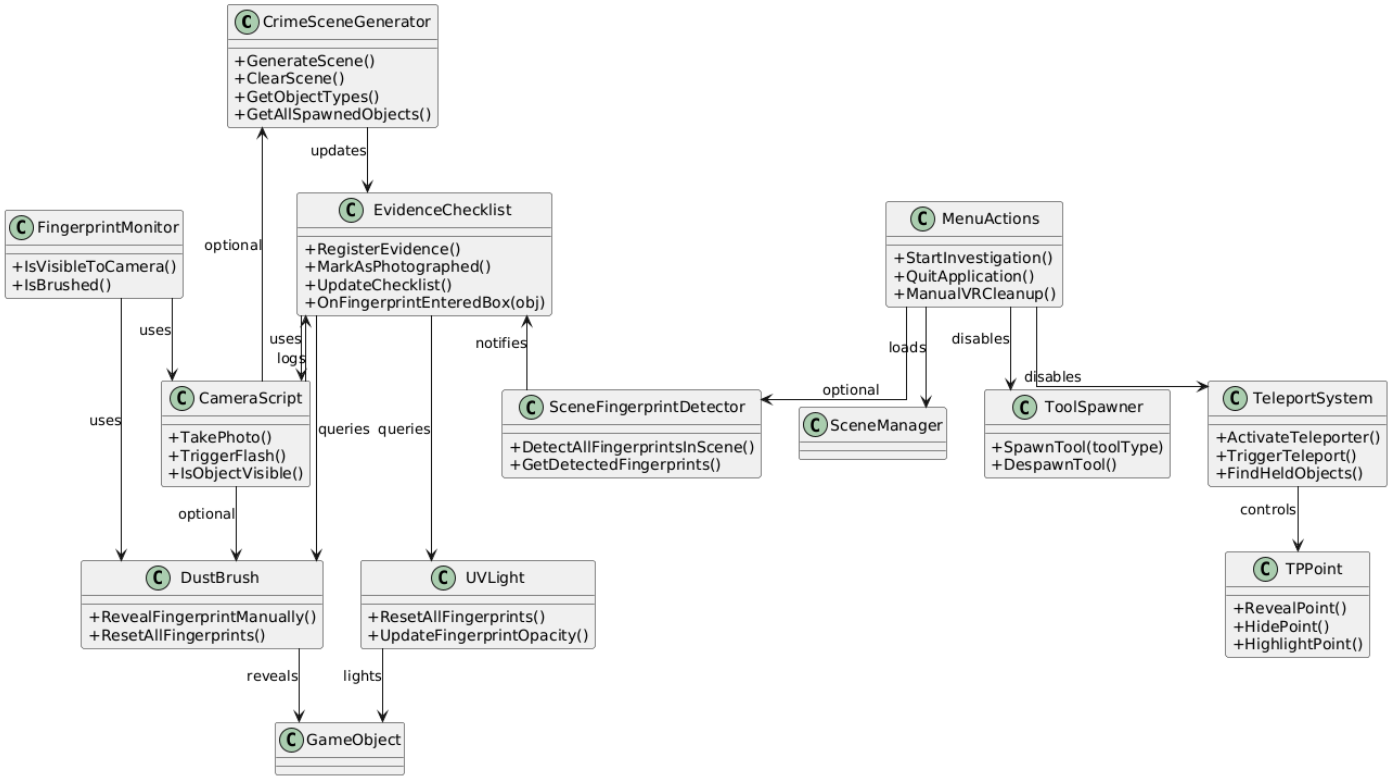
Emotional Distress:

As we will be dealing with real-life crime scenes and scenarios, I will need to take into consideration how these environments will affect people, and take the necessary steps to ensure they have support.

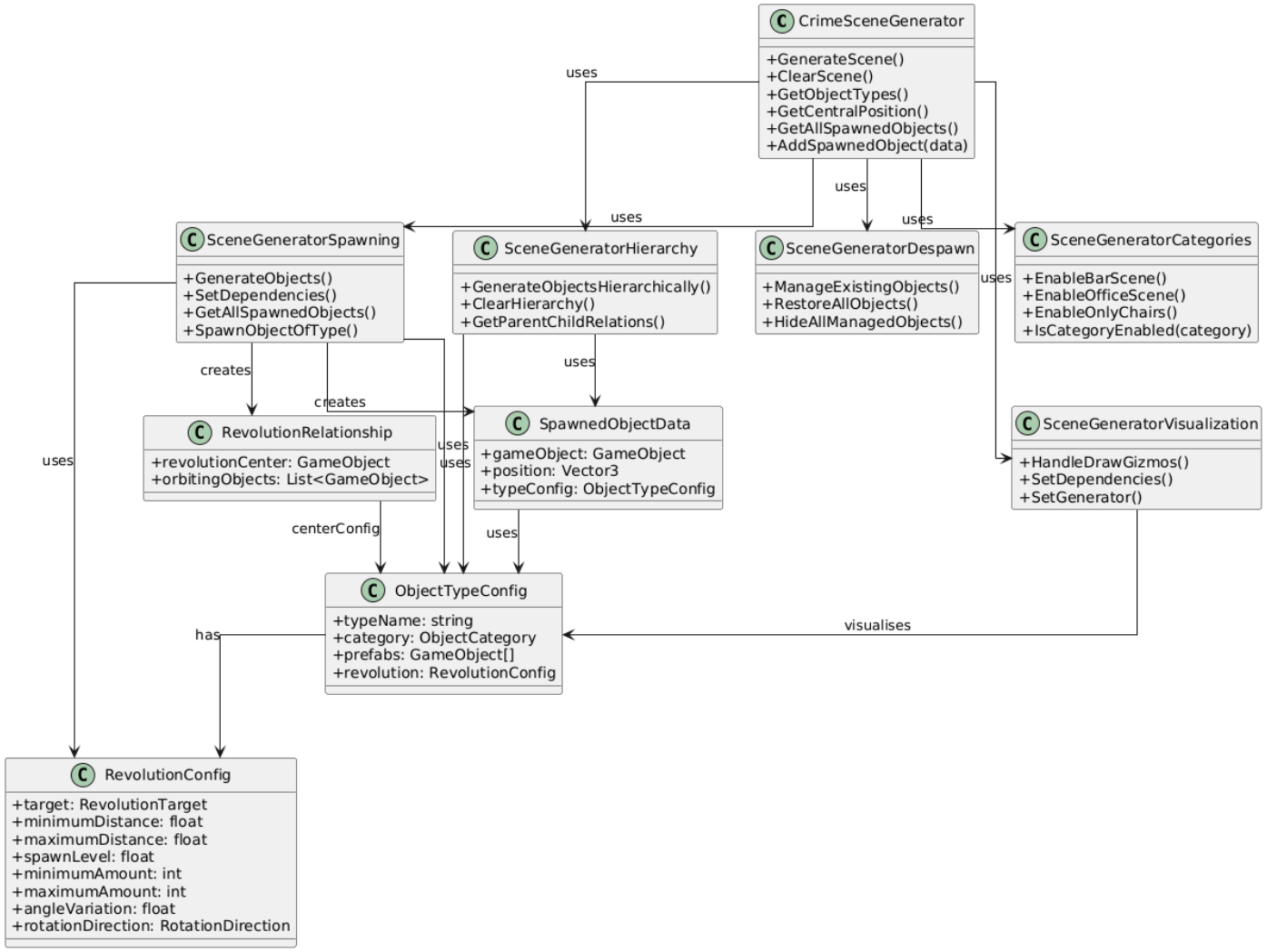
Class Diagrams

I created these class diagrams in PlantUML by manually examining the source code for each system and mapping out the relationships between key components. This gave me a structured overview of how core classes like CrimeSceneGenerator, EvidenceChecklist, and DexterAI connect with tool logic, evidence systems, and procedural workflows. By splitting the procedural generation and gameplay toolsets into separate vertical layouts, the diagrams keep things readable while clearly showing which scripts handle which responsibilities. They offer a clean, accurate snapshot of the architecture, ideal for both documentation and explaining how everything fits together.

Version 1.0



Version 1.1


CrimeSceneGenerator
+GenerateScene()
+ClearScene()
+GetObjectTypes()
+GetCentralPosition()
+GetAllSpawnedObjects()
+AddSpawnedObject(data)
uses
uses
uses
uses
SceneGeneratorSpawning
+GenerateObjects()
+SetDependencies()
+GetAllSpawnedObjects()
+SpawnObjectOfType()
SceneGeneratorHierarchy
+GenerateObjectsHierarchically()
+ClearHierarchy()
+GetParentChildRelations()
SceneGeneratorDespawn
+ManageExistingObjects()
+RestoreAllObjects()
+HideAllManagedObjects()
SceneGeneratorCategories
+EnableBarScene()
+EnableOfficeScene()
+EnableOnlyChairs()
+IsCategoryEnabled(category)
uses
creates
creates
uses
RevolutionRelationship
+revolutionCenter: GameObject
+orbitingObjects: List<GameObject>
SpawnedObjectData
+gameObject: GameObject
+position: Vector3
+typeConfig: ObjectTypeConfig
SceneGeneratorVisualization
+HandleDrawGizmos()
+SetDependencies()
+SetGenerator()
uses
uses
uses
centerConfig
uses
ObjectTypeConfig
+typeName: string
+category: ObjectCategory
+prefabs: GameObject[]
+revolution: RevolutionConfig
visualises
has
RevolutionConfig
+target: RevolutionTarget
+minimumDistance: float
+maximumDistance: float
+spawnLevel: float
+minimumAmount: int
+maximumAmount: int
+angleVariation: float
+rotationDirection: RotationDirection