A Totally True Crime Story

Digital Prototypes

The Problems to Solve

Firstly, we broke our game down into the following major problems that require solutions:

- Managing Cutscenes;
- Handling ‘Evidence’, (Clues, Statements, and Corroborations);
- How to display all this information (UI);
- How does the player control the game itself;

Cutscene Manager - Initial System

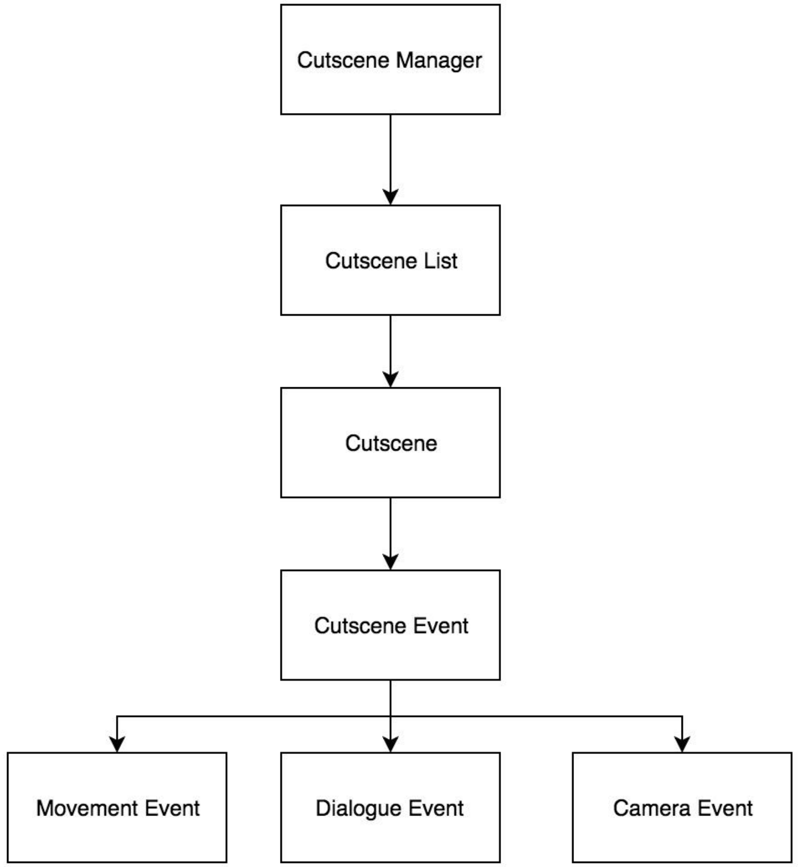
- Started with parsing a separate text file for cutscene instructions
- Pros: Easy to understand and read like a script.
- Cons: Added an extra layer of separation from unity.
- Vulnerable to user error.

Cutscene Manager - System 2

- Similar to the final version, except with a more complicated system of passing variables.
- Multiple bools passed around in order to monitor the completion of different functions between different scripts
- Slight use of a coroutine, but implemented poorly and no nested coroutines
- Pros: all in unity, controlled through the editor
- Cons: convoluted, required a system of different scripts all tied together in weird ways, has multiple crash states and hard to follow.

Cutscene Management - Final System

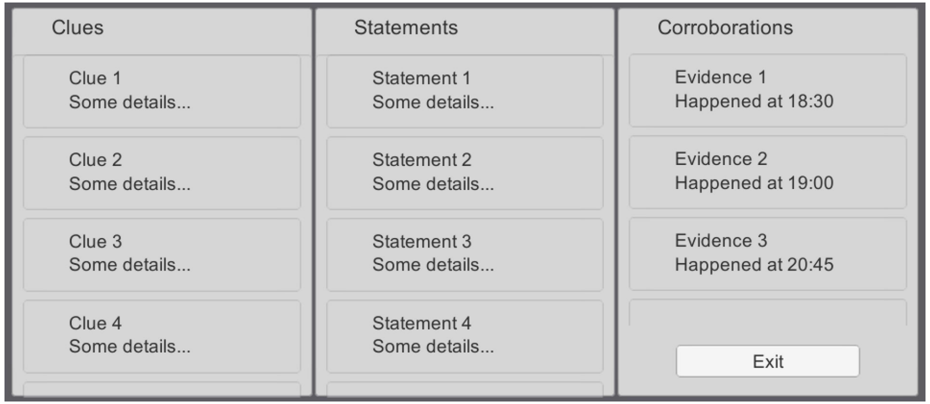
- 5 tiered system
- Each layer is a list of the lower tier
- Nested coroutines allow for a more stable system that’s easier to understand.
- Pros: still all in editor, much simpler, more stable
- Cons: currently has a small amount of implementation, camera zoom and movement is not tied together



Player Control - Initial System

- Creating the map.
- Navmeshing the map.
- Problems with creating/navmeshing the map.
- Making a detective object.
- Using the navmesh to help the detective object move using code.
- Detective changing colour when clicked.
- Making multiple detectives selectable by holding down an button and clicking on them.
- Creating an click n drag selection box using paint.net to create an 3x3 box then using the sprite editor in unity to finish it.
- Using the now made selection box to select multiple detectives at once using a click n drag method.

UI



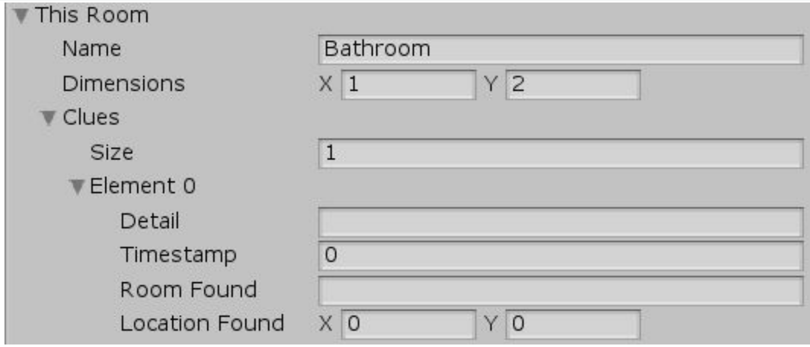
Clues	Statements	Corroborations
Clue 1 Some details...	Statement 1 Some details...	Evidence 1 Happened at 18:30
Clue 2 Some details...	Statement 2 Some details...	Evidence 2 Happened at 19:00
Clue 3 Some details...	Statement 3 Some details...	Evidence 3 Happened at 20:45
Clue 4 Some details...	Statement 4 Some details...	

- Started with making a basic system for the notebook.
- Designed the notebook pages and the timeline separately.
- Afterwards, I recreated the UI design from our paper prototype.
- Merged the notebook into one page with tabs.
- Added a corroboration menu to fix errors with switching tabs.
- Next step is to add feedback loops to the corroboration system.

Evidence Management - Problems

- Need to easily add evidence items into a level.
- Three major ‘types’ of evidence that need to be modeled:
    - Clue;
    - Statement;
    - Corroboration;
- One of which needs to reference other ‘Evidence’ objects.
- Needs to be flexible.

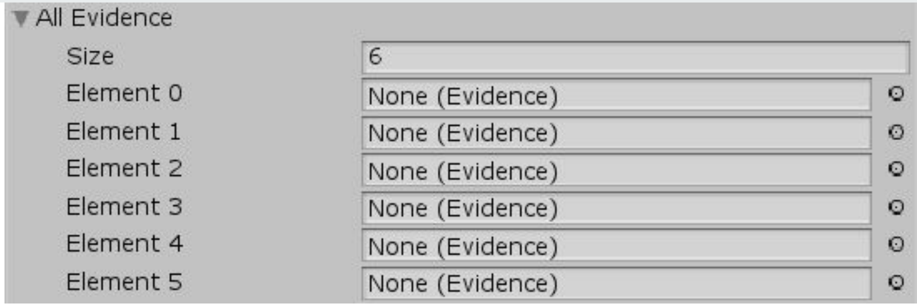
Evidence Management - System 1



This Room				
Name	Bathroom			
Dimensions	X	1	Y	2
Clues				
Size	1			
Element 0				
Detail				
Timestamp	0			
Room Found				
Location Found	X	0	Y	0

- Made use of structs to represent the three 'Evidence' types.
- Made use of C# Lists on rooms/witnesses to contain and organise clues/statements respectively.
- A simple hard-coded timer used to push back the information as it is 'Discovered'.
- Pros:
  - Very simple and usable;
  - Instantly formatted for alterations in the inspector;
  - Light and efficient;
- Cons:
  - Too lightweight? Further development made difficult;
  - Corroboration logic would involve the creation of objects anyways - defeats the purpose of using structs;

Evidence Management - System 2

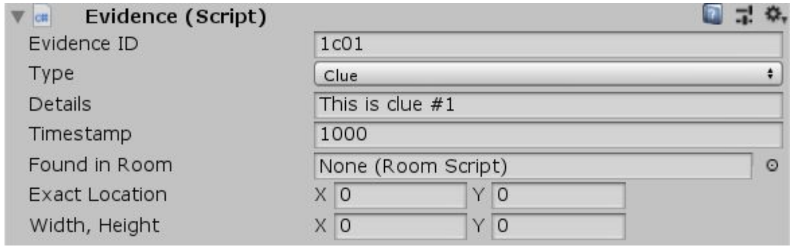


▼ All Evidence

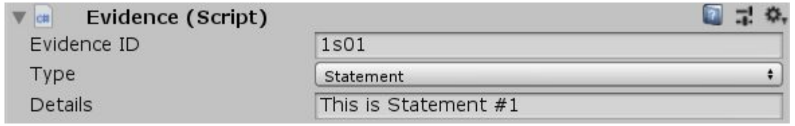
Size	6
Element 0	None (Evidence)
Element 1	None (Evidence)
Element 2	None (Evidence)
Element 3	None (Evidence)
Element 4	None (Evidence)
Element 5	None (Evidence)

- A base ‘Evidence’ class with the three evidence types represented by child-classes.
- Using coroutines to handle the timers tied to searching/interrogating.
- Began to use a Game Manager to handle the following:
  - References to detectives, rooms, and witnesses;
  - List of all ‘Evidence’ objects in the level;
  - Search/Interrogate and Corroboration logic;
- Pros:
  - Much more object-oriented;
  - Less abstract;
- Cons:
  - Very inefficient - don’t want to loop through long C# Lists;
  - A bit finnicky to add items into a level. Evidence exists all on the Game Manager;

Evidence Management - Final System

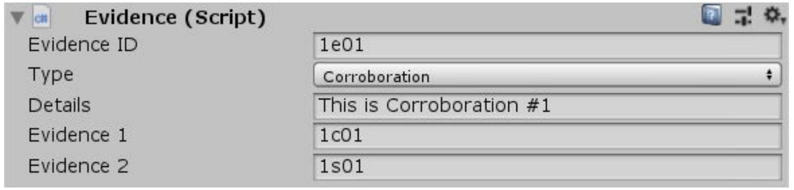


Evidence ID	1c01
Type	Clue
Details	This is clue #1
Timestamp	1000
Found in Room	None (Room Script)
Exact Location	X 0 Y 0
Width, Height	X 0 Y 0



Evidence ID	1s01
Type	Statement
Details	This is Statement #1

- One Evidence class with an enumerator defining type;
  - Enumerators and Inspector editing;
  - Added a fourth type of evidence, nullEvidence, to define a failed corroboration.
- The Game Manager now gets references to rooms dynamically;
- An EvidenceHandler class manages:
  - Lists of references to Evidence (Dynamically collected);
  - Collates 'found' evidence;
  - Handles corroboration;
- Added use of an InputManager;



Evidence ID	1e01
Type	Corroboration
Details	This is Corroboration #1
Evidence 1	1c01
Evidence 2	1s01

What’s Next?

- ‘Timeline Solution’ system;
- Interrogation math/balancing;
- Develop individual detectives;
- Peripheral systems - Menu, Settings, etc;
- Music & Aesthetics;
- Building the levels themselves;
- For the most part, story is complete;
- Feedback Loops galore;