This project was built in Unreal Engine5 with a focus on **modular systems** and **dynamic event management** to create an immersive and replayable experience. With easy to **expand systems**.

Below, you'll find an overview of the core systems and mechanics that bring this haunting world to life.

Play as **Elliot**, a man trapped in a loop of fractured reflections and fragile cause-and-effect. One moment, you're standing in a rotting house where mirrors weep memories. The next, you're lost in a forest where snowflakes fall upward and butterflies shatter like glass. Time splinters. Choices ripple.

Each mirror shows a different version of your past—some you remember, some you swear never happened. Behind every reflection hides a truth... or a lie. Your smallest action sends cracks through reality, changing rooms, people, even the rules of the world.

You begin to suspect you're not just reliving time—you're rewriting it.
But the more you change, the less you recognize yourself.

Time loops. Mirror worlds. A butterfly flapping its wings in your mind.
Will you escape the echo, or become another reflection, lost forever?

Core Systems Overview

Interaction System

`IInteract` - interacting with door, keys, lever, buttons and other interactable objects. `ILightSource` - Used to run "`FlickerLight()`" (flickering effect) and "`DestroyBulb()`" (On encounter with ghost the first time, triggered.) `IPlayer` - Used to trigger functions inside `PlayerCharacter` script like `Hallucination()` function, which enables `PostProcess()` component.

Event Manager System

`EventManagment` GameObject+Script is used to trigger/manage events

EventsEnumeration

- LightFlicker
- Halucinations
- BlackOut
- SoundPlay
- ShowMessage
- JumpScareEvent
- DisablePlayerTorch
- Handshake (to link two scripts together, upon contact)
- Teleport (for creating optical illusions and switching levels)

Each Instance of `EventManagerSystem` of has, variables that can be edited based on instance in editor.

- `EventIndex` (Events Enumeration datatype) - this one decides which main thing the particular EventObject would trigger before getting destroyed.
- `SendMessageToo?` (bool)
- `PlaySoundAsWell?` (bool)
- `DisablePlayerLights?` (bool) and `DisableLightsFor` (float)
- `HallucinationsAsWell?` (bool)

based on which bool is active, aside from the Event in `EventIndex` in getting triggered you can send/show/trigger multiple functions based on use-case. (like at one place i might desire to - show an message+sound), in one i might want (sound+hallucination+lightflicker) - so according to the scene dynamic need we can

`OnBeginOverlap()` when condition `ActorHasTag("Player")` is met, triggers the (Event in `EventIndex`) & Other Events based on which bool active

User Interface System

Functions List

- `UILoadChapter(Int LevelIndex, Int Duration)` - This function is responsible for showing opening text of every new chapter
- `UIMessageDisplay(Text Text)` - responsible for showing dialogues and informative texts (ex-picked up Kitch key, You need ABC to open this door.)
- `UIDisplayDiary(Text Title, Text Entry)` - Trigged when player interacts with an DiaryActor/NotesActor/etc - to display it's content.
- `UIUnDisplayDiary()`
- `UINoobsGuide()` - Displays UI for which key is responsible for what action (RMB-Interact, X-DropItem, LMB - UVFlashlight)

PlayerCharacter Script & GameObject

Some Important Functionalities

- Responsible for taking keyboard inputs and running functionality relating to said function
- There's an `SphereCollider` attached that
  - `OnBeginOverlap()` - checks for any interactable object (`Does Object Implement Interface? == IInteract`), to give player heads-up w/ UINoobsGuide() for how to interact with the object.
  - When user press `InputActionKey` for `Interact`, we iterate through all overlapping objects to check for (`Does Object Implement Interface? == IInteract`) if yes, then Trigger `Interact(this)` function utilizing the `IInteract` interface
- Also there's an `postprocess` component attached to this, that is enabled every time we wish to display hallucinations and other effects.
- Two Spot Light Components
  - one with shorter cone angle (serves as flashlight)
  - other with bigger cone angle (at low illuminosity) (to give an general ambient illumination to surrounding) for an much better experience

Keys/Possessable PickUp System

The data of whether player possess an key/possessable item is stored in an Bool Array, whose indexes correspond to which is stored in an Enum (like 0=BedRoomKey 1=KitchenKey,..)

When `IInteract()` function is run, it updates `BoolArray` set value at `(int)EnumGameObject`

Infinite Hallways/Staircase

The illusion is pulled by teleporting the player character around. *These systems too work with Event Manager, blackouts, light flicker to disorient player and not to notice change when we swiftly teleport them in middle of it.*

In infinite hallway there are four event manager object, two two on each end, the reason for there being two of them is that in case after teleportation, player try to break the game by running backwards, the other event manager will teleport it to one of inner event manager anchor.

Collectible Mirror Object

There are mysterious collectible mirrors placed around, each mirror does something that can serve as clue for person to solving the puzzle to unlock the mystery. → Reversing Movements (Going forward, makes you go backward, left→right, etc) (multiplying input-axis value by -1) → Reversing Gravity (we play with character controller script gravity balues) → Color Vision ↔ Black-White (playing with post-processing volume attached to player-character) → Giving Something leads to something taken away from you Wrapping the reality, reversing the world in ways. Each mirror has its own mystery

Mirror

Mirror is computationally expensive to render, so we tried avoiding using it wherever we can. In the initial scene where person sees ghost and an disoriented reflection of himself, we did that by creating an pawn class character with same mesh as character, when the player character steps into an event manager object, it performs handshake between player-character and that mirror image pawn for the input of one to be fed into the other.

I have tried to implement event driven systems instead of running things in **GameLoop Update** function, due to inefficiency of calling an function each frame.