SHADOWING

Stealth game
First-person puzzle

Video Link: https://youtu.be/SUTmdUah4-k

Made With

Unity®

Summary

Shadowing is a first-person stealth puzzle game set in a time-looping theater. Players repair temporal anomalies in the first playthrough, then return as their past self's unseen adversary, using recorded movements, shifting light, and environmental cover to create the very disruptions they once resolved.

Inspiration

The alternate-timeline encounter between two versions of Wraith inspired Shadowing's two-playthrough structure, in which the player eventually confronts the consequences of their own actions.



Titanfall 2

Its time-shifting level design inspired me to explore how temporal changes could reshape player movement and level progression.

Ekko — League of Legends

Ekko's time-reversal ability inspired my interest in replaying past actions and turning them into a core gameplay mechanic.





Story



A theater is caught in a time loop. Three acts—Arrival, Independence, and Departure—repeat endlessly, while the performers' movements become inexplicably distorted. Drawn to these anomalies, the player arrives as a time repairer, seeking to restore the performances.

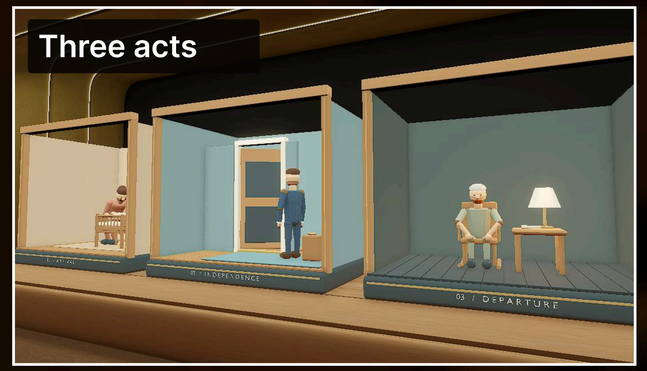
Later, the player returns from the future to prevent an impending fire. Without being seen by their past self, they must create the very temporal fractures they once repaired, drawing their earlier self into the theater. The anomalies become a distress signal across time, revealing a causal loop: the repairer is also the source of the disruption.

Game Design

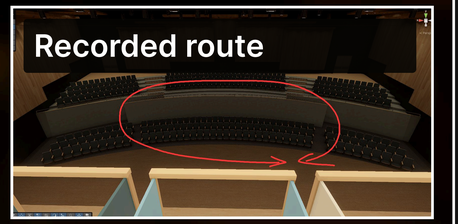
Repair the anomalies. Return to become their cause.

First playthrough

Repair three acts. Record your route and gaze for the next playthrough.

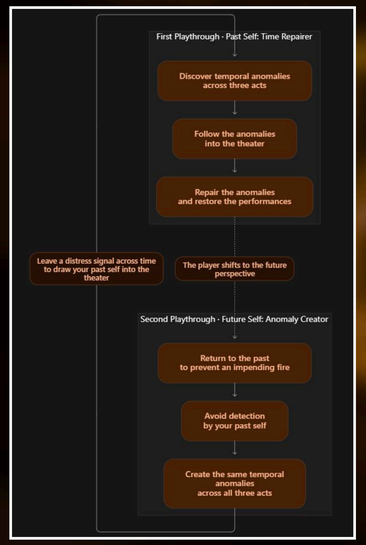






Temporal effects:
Freeze · Rewind · Frame jumps

Causal loop



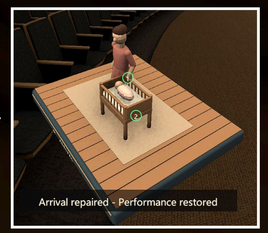
Move sideways to rotate · Look up / down to tilt



Observe



Adjust



Repair

Find the angle.
Click to repair.

Second playthrough

Recreate the anomalies you once repaired—while avoiding your past self.



Avoid the recorded gaze.



Detection meter rises.

Three rounds · One irregular spotlight





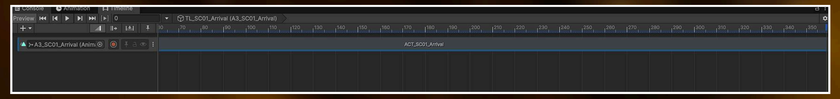
One light breaks the rhythm.
Stand still beneath it to create an anomaly.

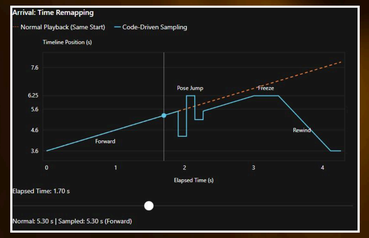
Programming

Timeline

Each act runs on Timeline.
Remap time to create anomalies.







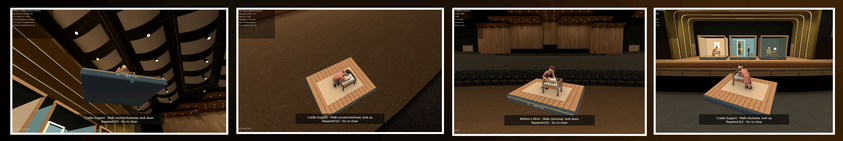

```
// Offline preview and validation call this on a temporary prefab instance.
public void EvaluateElapsed(double elapsed)
{
    if (!AcquireDirector()) return;
    CurrentSample = profile.GetTiming(director.duration).Sample(elapsed);
    director.time = CurrentSample.SourceTime;
    director.Evaluate();
}
```

Remap time → Evaluate pose

Freeze · Jump · Rewind

Spatial Interaction

Repair unlocks at the target viewing angle.



Pitch → tilt

Move → rotate

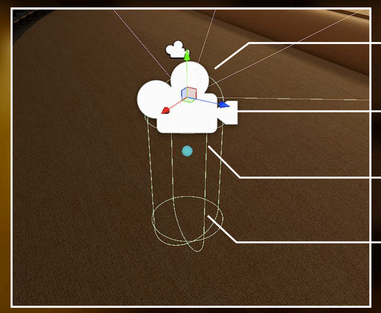
```
float targetVertical = -(_playerSimulation.Pitch - _entryPitch) * verticalDegree
_verticalAngle = Mathf.Lerp(_verticalAngle, targetVertical, blend);
_overlayCamera.transform.rotation = _presentationRotation;
_instance.transform.localRotation = Quaternion.AngleAxis(_verticalAngle, Vector
    * _initialViewRotation * Quaternion.AngleAxis(_horizontalAngle, Vector3.up)
```

```
            {
                if (_hasBearing) _accumulatedOrbitAngle += Mathf.DeltaAngle(_previousBearing, bearing);
                _previousBearing = bearing;
                _hasBearing = true;
            }
            else _hasBearing = false;
            float targetAngle = _accumulatedOrbitAngle * horizontalDegreesPerOrbitDegree;
            float blend = 1f - Mathf.Exp(-rotationSharpness * Time.deltaTime);
```

Replay System

First-playthrough data drives the past self's movement, posture, and gaze.

01 / Record player state




```
114         {
115             Vector3 position = sourcePlayer.transform.position;
116             var state = sourcePlayer.State;
117             var config = sourcePlayer.Config;
118             float viewHeight = Mathf.Lerp(config.CrouchingCameraHeight, config.StandingCameraHeight, state.TransitionProgress);
119             var flags = sourcePlayer.LastMotorResult.IsGrounded ? ReplayPoseFlags.Grounded : ReplayPoseFlags.None;
120             if (state.Posture == Posture.Crouching) flags |= ReplayPoseFlags.Crouching;
121             if (state.TargetPosture == Posture.Crouching) flags |= ReplayPoseFlags.TargetCrouching;
122             return new ReplayPose(tick, position.x, position.y, position.z, sourcePlayer.Yaw,
123                 sourcePlayer.Pitch, state.CurrentHeight, viewHeight, state.LeanOffset, flags);
```

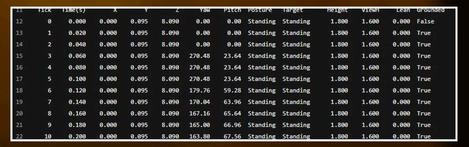
Capture movement, view, and posture after each fixed simulation tick.

02 / Save recording

Full-playthrough binary storage.
Developed with AI assistance.



Binary replay file



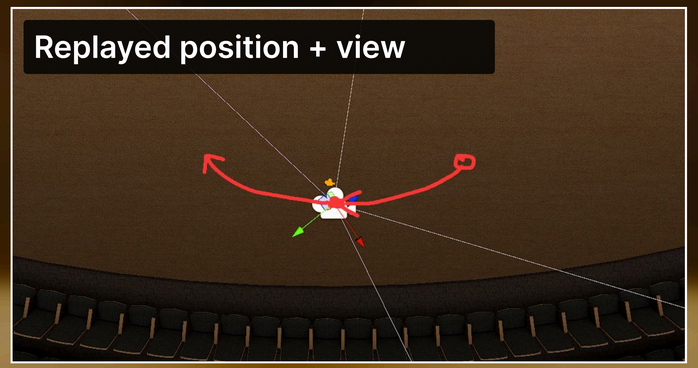
	Tick	Time(s)	X	Y	Z	Yaw	Pitch	Posture	Target	Height	View	Lean	Grounded
12	0	0.000	0.000	0.095	8.090	0.00	0.00	Standing	Standing	1.800	1.600	0.000	False
13	1	0.020	0.000	0.095	8.090	0.00	0.00	Standing	Standing	1.800	1.600	0.000	True
14	2	0.040	0.000	0.095	8.090	0.00	0.00	Standing	Standing	1.800	1.600	0.000	True
15	3	0.060	0.000	0.095	8.090	270.48	23.64	Standing	Standing	1.800	1.600	0.000	True
16	4	0.080	0.000	0.095	8.090	270.48	23.64	Standing	Standing	1.800	1.600	0.000	True
17	5	0.100	0.000	0.095	8.090	270.48	23.64	Standing	Standing	1.800	1.600	0.000	True
18	6	0.120	0.000	0.095	8.090	179.76	59.28	Standing	Standing	1.800	1.600	0.000	True
19	7	0.140	0.000	0.095	8.090	170.04	63.96	Standing	Standing	1.800	1.600	0.000	True
20	8	0.160	0.000	0.095	8.090	167.16	65.64	Standing	Standing	1.800	1.600	0.000	True
21	9	0.180	0.000	0.095	8.090	165.00	66.96	Standing	Standing	1.800	1.600	0.000	True
22	10	0.200	0.000	0.095	8.090	163.80	67.56	Standing	Standing	1.800	1.600	0.000	True

Decoded samples

Encode & write · AI-generated

```
    ReplayPose[] frames = ReplayTrackSimplifier.Simplify(copy);
    byte[] payload = ReplayCodec.Encode(frames);
    _writer.Write(ReplayCodec.ChunkMagic); _writer.Write(start); _writer.Writ
    _writer.Write(frames.Length); _writer.Write(payload.Length); _writer.Writ
    _writer.Write(payload); _writer.Flush();
    _chunks++; _end = end;
}
```

03 / Replay




```
{
    if (b.Tick <= a.Tick || tick <= a.Tick) return a;
    if (tick >= b.Tick) return b;
    float t = (float)((tick - a.Tick) / (b.Tick - a.Tick));
    return new ReplayPose((long)tick, Lerp(a.X, b.X, t), Lerp(a.Y, b.Y, t),
        Lerp(a.Z, b.Z, t), a.Yaw + AngleDelta(a.Yaw, b.Yaw) * t,
        Lerp(a.Pitch, b.Pitch, t), Lerp(a.Height, b.Height, t),
        Lerp(a.ViewHeight, b.ViewHeight, t), Lerp(a.Lean, b.Lean, t), a.Flags);
```

Interpolate between recorded samples for smooth movement and camera motion.

A1-A2-A3

Three iterations of my first game.



A1 / Exploring the Shadow

My first prototype let players record and replay a ten-second movement path. The shadow could only reproduce movement, limiting its interactions. This established the core idea, but the stealth gameplay needed more depth.

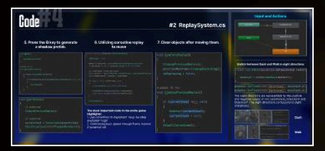




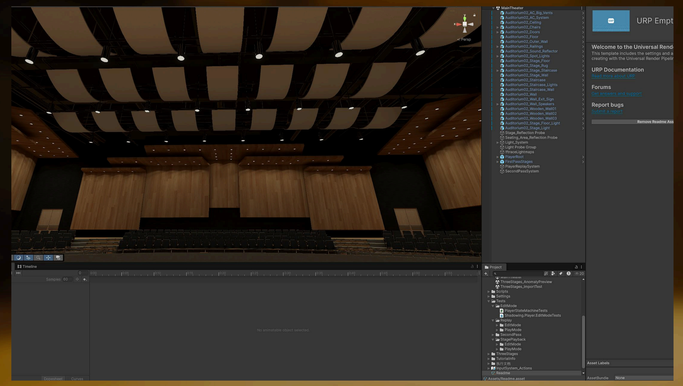
A2 / Developing the Gameplay

A2 turned the shadow into a distraction tool for top-down stealth puzzles. I added camera-obstruction transparency an explored Unity's testing tools. The key lesson was that technical progress alone did not give the game a distinct identity.













For the specifics of the A2 project, you can check out the game manual on my personal website.

A3 / Connecting Mechanic and Story

A3 connects replay with a causal loop: players first repair anomalies, then return to create them while avoiding their past self. The recording becomes both a gameplay threat and part of the story.

Ever Forward

Across nearly two years, this project has grown alongside me. A3 is still unfinished, but it reflects how my thinking has changed—from implementing individual mechanics to connecting systems, player experience, and narrative. I do not know whether I will return to it, but I will carry these lessons into whatever I make next.

Special thanks to Chen "HermLt" Ruiqi for the original gameplay blueprint.