Sunset Express — Technical Postmortem

Internal working / repository name: Coffin.

Project: Coffin — a two-player cooperative physics game **Engine / networking:** Unity 6000.0.79f1 · FishNet **Active period:** 2026-07-07 → 2026-08-19 (44 days) **Status:** Halted. The last working build lives on the `ragdoll_oncesi` branch.

---

1. What the game was

Two players carry a coffin together. The coffin is not a kinematic prop — it is a real rigidbody with mass, solved from forces applied by two separate clients. The entire design is built around this one object.

The corpse inside the coffin is part of the simulation too:

- it slides once a tilt threshold is crossed (`corpseTiltThreshold = 10°`),
- it ejects if the lid opens far enough (`lidOpenAngleThreshold = 45°`) and takes a hard enough impact (`lidImpactImpulseThreshold = 400`).

So players are not carrying a box — they are carrying a **shifting mass distribution inside a box**. On stairs, the corpse sliding backward makes the carry harder. That is not an animation; it is a direct consequence of the force balance.

The entire engineering burden of the project fits in one sentence: **a shared physics body, driven by input from two different clients, solved deterministically.**

---

2. Architecture: the deterministic tick pipeline

Problem

Networked physics is not a *physics* problem. It is an **ordering** problem.

By default, Unity's physics engine advances on its own fixed step, independent of the network layer. That does not work in multiplayer physics: the server and the client must solve the same input in the same order, on the same step. Otherwise the two machines diverge within seconds.

Solution

Simulation was handed entirely to the network layer:

- Project setting — `ProjectSettings/DynamicsManager.asset` → `m_SimulationMode: 2` (Script)
- Scene setting — `_physicsMode: 1` (TimeManager) in every scene

Unity now never steps on its own. The sole owner of the `Physics.Simulate()` call is FishNet's tick loop.

Order of operations inside a tick

#	Stage	What happens
1	Gather input	Local player's replicate data for that tick
2	Replicate	Forces are written to rigidbodies — nothing is solved yet
3	`Physics.Simulate()`	**One call, one owner. Every body solves at the same instant**
4	Reconcile	Compared against server state; rewound if it diverged
5	`PublishTransform`	The visual layer reads from physics state — never writes back

> **Worth flagging:** if the scene setting and the project setting disagree, you get a double simulation. The symptom looks like "occasional stutter," and tracking down the cause takes days. These two must be checked together.

The cost of this architecture

Every line that writes directly to a transform is forbidden. If non-physics code mutates a transform, that change is either lost during rewind or produces a different result on the two machines.

Only **five** places in the project publish transforms, and all of them go through the same one-way channel:

```
Assets/_Game/Scripts/Runtime/Player/PlayerController.cs
Assets/_Game/Scripts/Runtime/Coffin/CorpseSlide.cs
Assets/_Game/Scripts/Runtime/Obstacles/MovingPlatform.cs
Assets/_Game/Scripts/Runtime/Obstacles/SeesawBridge.cs
Assets/_Game/Scripts/Runtime/Networking/NetworkSceneDirector.cs
```

---

3. Force-based locomotion and the force band

The character is not a `CharacterController`. Every movement, walking included, comes from force applied to a rigidbody — because the same body is what carries the coffin.

The tension

A single movement-force value has to do two jobs at once:

- **too low** → the player cannot lift the coffin,
- **too high** → the player feels like they are sliding on ice while carrying nothing.

Unloaded feel and carrying capacity are bound to the same number.

Solution: a force band whose floor derives from mass

Instead of one constant, a two-ended band was defined (`Assets/_Game/Scripts/Runtime/Physics/PlayerProfile.cs`):

```
// floor that preserves unloaded feel
floorForce = MoveAccelPerTick × mass ÷ tickDelta      // MoveAccelPerTick = 0.8

// ceiling that provides carrying capacity
MoveMaxForce = 2800 N

// applied
F = clamp(desired, floorForce, MoveMaxForce)
```

Drop the ceiling and the coffin will not lift. Raise the floor and unloaded running turns slippery. Because the two are tunable independently, feel and capacity no longer hold each other hostage.

Carry-state coefficients

While carrying, the character's behavior changes through coefficients rather than animation:

Parameter	Value	Effect
`CarryJumpFactor`	0.65	Jump strength is reduced while carrying
`CarryIdleBrakeFactor`	0.45	Braking weakens at rest — the load drags you slightly
`grabBreakForce`	4500	The grip breaks at this force
`grabBreakDeviation`	0.30	Angular deviation tolerance — hold it crooked and it slips
`regrabCooldown`	0.5 s	Wait time before re-gripping after a break

As a break approaches, the player gets a three-stage warning at ratios `0.50` / `0.65` / `0.80`. Losing the grip is therefore not a surprise but a readable threshold.

---

4. The coffin: one shared body, two owners

The nastiest case in networked physics is two different clients applying force to the same rigidbody simultaneously.

Connection model

The coffin attaches to both players via `ConfigurableJoint` (`Assets/_Game/Scripts/Runtime/Physics/CoffinProfile.cs`):

Parameter	Value
`baseShellMass`	100
`jointLinearLimit`	0.08 m
`jointLinearLimitSpring`	15000
`jointLinearLimitDamper`	200
`maxAngularVelocity`	7
`yawDamping`	6

The coffin does not **stick** to the player; it hangs very stiffly over a very short distance. That distinction is critical: if it were stuck, two players' conflicting input would teleport the body instantly. Because it is spring-coupled, the conflict turns into force — and the players genuinely feel each other.

The synchronized-jump case

When both players jumped at the same moment, the sudden impulse loaded directly into the spring; the coffin launched and the grips broke.

The fix was **not** to reduce jump force, but to make the system more tolerant for a brief window:

Parameter	Value	Effect
`syncJumpBreakWindow`	0.40 s	How long the tolerance stays open
`syncJumpBreakForceMultiplier`	2×	Break threshold temporarily doubles
`syncJumpSpringMultiplier`	3×	Spring stiffens — impulse goes into carrying, not stretching

All three work together. Without the raised threshold the grip breaks; without the stiffer spring the coffin stretches the spring and snaps back. None of them sufficed alone.

---

5. The ragdoll pivot and three attempts

Context

On August 7 the game worked. v0.3 had just closed after the most intense stretch of the project. At that point the decision was made to convert the characters to active ragdolls.

The reasoning was sound: if carrying is already force-based, let the character itself be force-based too — arms reaching toward the coffin, staggering that is real. The problem was not the reasoning. It was **assuming this was compatible with the existing architecture.**

Attempt 1 — Active ragdoll rig from scratch

Branch: `ragdoll_v04` · Aug 7–18

Nine separate runtime classes were written:

```
Runtime/Ragdoll/HumanoidRagdollBuilder.cs
Runtime/Ragdoll/ActiveRagdollController.cs
Runtime/Ragdoll/ProceduralGaitPlanner.cs
Runtime/Ragdoll/RagdollContactSensor.cs
Runtime/Ragdoll/RagdollPoseSource.cs
Runtime/Ragdoll/RagdollPoseTeleport.cs
Runtime/Ragdoll/RagdollRig.cs
Runtime/Physics/RagdollProfile.cs
Runtime/Debugging/RagdollSpikeHarness.cs
```

It never settled deterministically. Keeping a structure with dozens of joints rewind-compatible is a **categorically different** problem from keeping one rigidbody compatible — and that problem had not even been *stated* before work began on solving it.

Outcome: abandoned.

Attempt 2 — Same approach, more tuning

Same branch.

A profile asset (`RagdollProfile`) and a debug harness (`RagdollSpikeHarness`) were added. In other words: rather than question the architecture, I **expanded the tuning surface.** This is the most efficient way to keep a broken approach broken for longer.

Outcome: abandoned.

Attempt 3 — Rebuild from a known-good tutorial

Branch: `master` · Aug 18–19

My own code was set aside in favor of following a known-working reference implementation:

```
Editor/ActiveRagdollBuilder.cs
Editor/TutorialRagdollTestSceneBuilder.cs
Runtime/Physics/TutorialRagdollProfile.cs
Runtime/Ragdoll/ActiveRagdollBody.cs
Runtime/Ragdoll/ActiveRagdollCollisionIgnore.cs
Runtime/Ragdoll/ActiveRagdollPoseLink.cs
Runtime/Ragdoll/ActiveRagdollTestCameraBinder.cs
Runtime/Ragdoll/ActiveRagdollTimeManagerDriver.cs
```

Two days of work, on August 18 and 19. Then nothing.

Outcome: project halted.

---

6. Findings

The core finding

What all three attempts share is that none of them asked the same question:

> **Is this feature compatible with the constraint holding the game up?**

Determinism was not a preference in this project; it was a load-bearing wall. Weeks had gone into keeping a single rigidbody deterministic inside the tick pipeline. Adding dozens of joints was not carrying that wall — it was drilling through it. And I tried it three times, each time with more code.

The real mistake was not technical but a matter of **sequencing**: I built a feature that invalidated a working system on top of that system, with no rollback plan. The `ragdoll_oncesi` branch exists because I eventually had to go back — but by then two weeks were gone.

What held

- **Single-owner simulation.** Handing `Physics.Simulate()` to the tick loop reduced the source of every later desync bug to one place.
- **One-way transform channel.** Five publishers, zero write-backs. That is why rewind was reliable.
- **Derived bounds.** Computing the force floor from mass decoupled "feel" from "capacity."
- **Readable thresholds.** The three-stage break warning made the physics learnable for the player.

What broke

- **Pivoting without validating the constraint.** Ragdoll compatibility with determinism was never tested; the work went straight to full implementation.
- **The reflex to tune instead of diagnose.** The second attempt added a profile and a harness; neither investigated the cause.
- **Creating the rollback point after the fact.** `ragdoll_oncesi` was branched *after* things went sideways, not before the pivot.
- **Pivoting straight off a peak.** Attempting an architectural leap immediately after the most intense stretch of the project meant making the riskiest decision at the most exhausted moment.

How I would do it today

The ragdoll would be tried as a spike — in a scene completely separate from the main codebase, answering exactly one question:

> Do two clients produce the same pose after rewind, on an articulated body?

If the answer were no, the loss would have been two days, not two weeks.

Closing

```
git checkout ragdoll_oncesi
```

A working cooperative physics game.

That branch does not point at the project's *best* state — it points at its state **immediately before the pivot.** That these are the same commit is the finding of this postmortem.