Ghost Engine Phase 3 Roadmap
CodeLaunch (April 2026) & Beyond

Project Lead Draft • Jan 12, 2026

1) Reframe after CES 2026: why Ghost Engine Phase 3 is timely

CES is effectively telling the same story from two ends:

- **Compute is getting radically cheaper/faster.** NVIDIA's Rubin platform is positioned around "extreme codesign" across multiple chips and claims **up to 10× lower inference token cost** vs. Blackwell, plus new infrastructure for **agentic AI context memory**.
- **AI is becoming ambient across consumer "surfaces."** Google previewed **Gemini for Google TV** features (richer query responses, "deep dives," photo search/remixing, and generating media directly on TV), with expansion across brands and even projectors.

What that means for Ghost Engine: the bottleneck shifts away from *getting a model to output something* and toward **turning outputs into deterministic, debuggable, portable interactive experiences** across engines, devices, and media formats.

2) North Star (Phase 3 thesis)

Ghost Engine OSS = **open orchestration + determinism + provenance + eval + adapters**.

If we nail that, "moving target AI" becomes a tailwind: every new model/API is just another adapter; the kernel stays the stable layer.

The differentiator in one line

Ghost Engine is the open-source interoperability kernel that turns Large World Model outputs into playable worlds.

3) Phase 3 product definition: protocol + platform

3.1 The protocol (Ghost Protocol v0.1)

A small, opinionated set of schemas/contracts that remain stable even as generators change:

- **WorldState + Tick + Events** (authoritative simulation core)

- **AssetManifest** (what assets exist, where they came from, hashes, licenses)
- **InteractionSpec** (colliders, triggers, interactables, nav graph, spawn graph)
- **ProvenanceSpec** (prompt, seed, model/provider version, build hash)
- **EvalSpec + EvalReport** (regression suite + metrics over time)
- **AdapterSpec** (how providers expose generation + exports)

3.2 The platform (reference implementations)

- **Reference Kernel** (deterministic simulation & state updates)
- **Orchestration Fabric** (job scheduling, caching, policy, retries, cost controls)
- **Adapter SDK** (WorldProvider interface + tooling)
- **Reference Runtimes**
- **GE DOOM** = determinism/debug runtime (ASCII or lightweight)
- **Minimal 3D Viewer** = “proof of portability” runtime (not a full engine)
- **Transmedia exporters** (glTF/GLB, USD-lite, video flythrough, web embeds)

4) CodeLaunch April demo: what we must be able to show in 2 minutes

Generate → Import → Interactive

- Prompt/preset world request
- Orchestrator calls provider (Marble first; SampleProvider always works offline)
- Kernel ingests exports → builds collision + nav + triggers
- Run as a playable loop (GE DOOM + optional 3D viewer)
- Re-run with same seed → same level (determinism)
- Show provenance + cached rebuild (“this build used world hash X, prompt Y, seed Z”)
- Run the eval suite; show pass/fail deltas

5) Roadmap: Jan → CodeLaunch (April) → Beyond

5.1 Now → April (12-week plan)

Week 1 — Repo + Protocol

- Monorepo layout; “kernel-first” README; licenses & contribution docs
- Ghost Protocol v0.1 schemas (JSON + TS types) + versioning rules

- CLI skeleton: `ge new`, `ge run`, `ge replay`, `ge export`

Week 2 — Deterministic Kernel MVP

- WorldState, Tick loop, deterministic RNG, event log
- Snapshot/restore + replay
- Unit tests for determinism (seeded runs)

Week 3 — GE DOOM as Reference Runtime

- Refactor Phase 1 demo to consume *only* WorldState + Events
- "Determinism debug view": render from authoritative state
- Capture/replay from log

Week 4 — Adapter SDK + SampleProvider

- Define `WorldProvider` interface: `generate()`, `fetchAssets()`, `getColliders()`, `metadata()`
- Build SampleProvider with bundled assets so anyone can run demo offline
- Add caching (content-addressed) + provenance capture

Week 5 — MarbleProvider Integration

- Adapter that requests a world, tracks job, downloads exports
- Normalize exports into AssetManifest + InteractionSpec inputs

Week 6 — Ingestion → Interaction

- Collider import → collision map
- Auto navmesh (coarse is fine)
- Spawn graph + pickup/trigger placement heuristics (rule-based first)
- Minimal AI agent loop (patrol/chase) driven by nav graph

Week 7 — Minimal 3D Viewer Runtime

- Simple runtime (Three.js/Godot/Unity sandbox) to prove engine-agnostic state
- Reads same WorldState; shows colliders/nav; basic interaction

Week 8 — Orchestration Fabric

- Local-first orchestrator (queue, retries, concurrency)
- Optional Cloud adapter stubs (Cloud Run/AWS) without making them "the product"

Week 9 — Eval Harness v0

- Regression suite: determinism, collision sanity, nav connectivity, gameplay metrics
- Baseline snapshots per provider + version
- CI: run eval on PRs

Week 10 — Transmedia Export v0

- glTF/GLB export of meshes + collision layers
- “Record” exporter: scripted camera flythrough (video)

Week 11 — Packaging + Docs

- Developer quickstart; adapter authoring guide
- Example worlds + demo scripts
- Public release: `v0.1.0`

Week 12 — CodeLaunch Prep

- 8-slide pitch deck
- 30-second demo video
- Live demo script + fallback recordings

5.2 Beyond April (next 6–12 months)

Q2–Q3 2026 — “Adapter flywheel”

- Add 3–5 adapters (open-source generators, photogrammetry, video-to-world, etc.)
- Unity + Unreal “thin plugins” (import + state bridge, not engine replacement)
- Better policy layer: safety, cost ceilings, tenancy
- Multi-world composition (“WorldGraph”) + streaming chunks

Q4 2026 — “Omnimedia platform”

- Film/virtual production export packages (USD pipelines, shot lists)
- VR/AR export templates (WebXR + engine integrations)
- Team workspace: shared caches, provenance ledger, eval dashboards (hosted)

2027+ — “Ghost Engine Cloud”

- Managed orchestration + caching + evaluation as a service
- Marketplace for adapters/eval packs/exporters
- Enterprise deployments with compliance and provenance guarantees

6) Guardrails: how we avoid drifting again

- **Kernel-first definition of done:** every new feature must touch WorldState/Tick/Events or Eval/Provenance.
- **Rendering is a non-goal:** we prove portability with a minimal viewer, not a “better engine.”
- **Cloud is optional:** ship local-first demos; cloud is an adapter.
- **Adapters must be swappable:** Marble is marquee, but never a dependency.

7) Risks & mitigations (Phase 3)

- **Vendor lock-in (Marble/API changes):** SampleProvider always ships; adapters are behind interfaces.
- **Scope creep (becoming a game engine):** GE DOOM is a reference runtime only.
- **IP/licensing ambiguity in generated assets:** provenance + manifest fields for license/usage + policy hooks.
- **Performance:** start coarse (colliders/nav), optimize after demo.

References (public)

- NVIDIA Rubin platform press release (Jan 5, 2026)
- Google “Gemini for Google TV” CES 2026 preview
- Epson press release on Google TV with Gemini on projectors (CES 2026)