



Marble Worlds - From Virtual Generati... File

You have access to the GitHub repo and to the Dropbox folder 'https://www.dropbox.com/scl/fo/fzizf9vu60i4lfb290tto/AMlBzqVNlJrQ3sZuESvzJk?rlkey=4s72vf4yajgb04u8p23zn42te&st=dnv9tmpk&dl=0'. We've completed the Phase1 ASCII experiment and have been working with Antigravity on a Phase 2 3D version. But, in light of recent AI developments such as https://marble.worldlabs.ai API, I'm wondering if we need to consider a slight pivot for the CodeLaunch Ghost Engine OSS submission? Claude did a deep dive on Marble plus other production pipeline implementations for transmedia and game development.

Based on a review of the Ghost Engine project and the recently released Marble platform, here's an assessment of the current state and recommendations for the CodeLaunch OSS submission:

Current Ghost Engine status

- **Phase 1 ASCII build** – the current engine generates deterministic Doom-like ASCII levels, exposes an HTTP API (`POST /generate-scene` , `GET /status/:id`) and includes tooling such as `ge-fetch` to pull a generated `levelSpec` and launch a local "GE Doom" executable. The demo report notes that replay determinism works, but some UI buttons are missing and manual port management is still required for the orchestrator/worker microservices .
- **Phase 2 plan** – Antigravity is working on a 3D version. Internal instructions outline adding DynamoDB persistence and CloudWatch observability, then implementing CI/CD for ECS

deployment . The goal is to keep the kernel simple: an API orchestrator dispatches generation jobs to a worker which returns a `levelSpec` that the client engine can render.

- **Missing features** – The engine currently handles simple procedural scene generation. 3D world creation, high-fidelity rendering and complex asset pipelines are not yet implemented.

Marble's capabilities (Nov 2025 update)

- Marble is a **multimodal world model** that can generate fully navigable 3D environments from text, images, video or layout sketches. It can edit and expand worlds and compose multiple worlds into large traversable spaces .
- It exports worlds as **Gaussian splats** (.ply/.spz) or **high-quality meshes** with accompanying low-poly collider meshes, making them compatible with Unity, Unreal Engine, Godot and Web-based renderers like SparkJS . Video exports with enhanced details are also supported .
- Although Marble is powerful, it is **not a game engine**; it focuses on environment generation. Interactivity (picking up objects, physics, gameplay) requires exporting assets into a separate engine . Editing is generative rather than object-level; there is no native runtime physics or scripting .
- Marble is now generally available via a freemium API (launched Nov 12, 2025) and includes a **Composer mode** that lets users assemble multiple worlds into larger experiences .

Implications for Ghost Engine

1. **Duplication of effort** – Building a proprietary 3D world generator would replicate features that Marble already provides at high fidelity. Marble's recent general-availability release shows rapid progress in world-scale generation , suggesting that small teams will struggle to match its quality or pace.
2. **Focus on differentiators** – Ghost Engine's unique value lies in deterministic scene generation, API-driven orchestration, and potential integration of gameplay logic. Marble does not handle interactive physics or game logic , so there is still a space for an open-source engine that layers interactivity, AI agents and procedural gameplay on top of generated environments.
3. **Opportunity for synergy** – Instead of building a 3D generator from scratch, Ghost Engine could become an **orchestrator** that calls Marble's API for environment generation, then automatically converts the exported splats/meshes into the Ghost Engine runtime. This would let the team focus on:
 - deterministic gameplay systems, agent AI and scripting,

- cross-media export tools (e.g., integrating with transmedia pipelines, mocap, VR/AR frameworks),
- wrappers and plugins for popular engines (Unity, Unreal, Godot) that apply Ghost Engine's logic to Marble-generated assets.

4. Consider licensing and openness – Marble is a closed, commercial platform. Any pivot should evaluate its API terms and costs. For an OSS submission, providing a modular “world-generator” interface that can swap between Marble, open-source alternatives or in-house generators may avoid vendor lock-in and keep the project open.

Recommendation

For the CodeLaunch submission, a slight pivot that **embraces Marble rather than competing with it** makes strategic sense. Instead of trying to build a competing 3D world generator, position Ghost Engine as an **open-source orchestrator and runtime** for AI-generated worlds. The engine could:

- Use Marble's API to generate high-fidelity environments from prompts or images, then automatically import the exported splats/meshes and collider meshes.
- Provide deterministic gameplay layers, AI character behavior and physics that Marble lacks
- Offer tooling for multi-world composition, level replay, and transmedia production (e.g., export to VR/AR, film pipelines, game engines).
- Maintain pluggability so that other 3D generators (open-source or future models) can be integrated.

This pivot leverages Marble's cutting-edge environment generation while preserving the Ghost Engine's differentiators and allowing the project to innovate in interactive transmedia experiences.



analyze the files in our current ChatGPT 'Ghost Engine for DevPost & CodeLaunch' project folder. I think we've somewhat deviated from Ghost Engine OSS' vision a bit.

Thought for 28m 25s >

I went through **everything that's currently uploaded into this ChatGPT "Ghost Engine for DevPost & CodeLaunch" project folder** (the files you see in this project's file list). Big picture: the folder contains a *very strong vision* and a *ton of contest-driven infra planning*, but comparatively little that "forces" the OSS kernel vision to stay central.

Below is what's in here, what it implies, and where the drift is coming from.

What's in the project folder right now

1) Vision + "what Ghost Engine is supposed to be"

- 0 Ghost Engine - updated roadmap for CodeLaunch.pdf

This is basically the "north star" restatement: Ghost Engine as a **hybrid framework** that plugs generative AI modules into a **deterministic simulation engine** (world state + physics stays authoritative; AI embellishes visuals/audio), with a layered architecture:

- **Simulation kernel** (world state + physics)
- **Imagination layer** (AI image/video/audio generation)
- **Orchestration fabric** (scheduling, policies, caching)

It also references a higher-level agent layer.

0 Ghost Engine - updated roadma...

This document is the clearest "OSS vision anchor" in the *current* folder.

2) Hackathon/infra gravity (why the project started feeling like "Cloud Run microservices")

- DevPost - Cloud Run Hackathon Rules.pdf

It strongly nudges architectures that maximize DevPost scoring—especially "bonus points" for orchestrating **multiple Cloud Run services** (service + worker pool), and using Google foundation models/AI tools.

DevPost - Cloud Run Hackathon R...

That incentive structure alone explains a lot of the orchestrator/worker emphasis.

- Google Documentation Links.txt

Just links to Cloud Run + Gemini API + ADK docs.

Google Documentation Links

- 0-1 ChatGPT convos2.zip + 0-2 Grok convos1.zip + **the postmortem bundle zip**

These are *mostly* operational scaffolding: repo setup, deployment steps, IAM/WIF, CI/CD, contest checklist/postmortem, and even exploration of “next contest” paths (e.g., Polkadot). Useful, but they push the project “center of mass” toward infra/contest compliance instead of kernel/product.

Important detail from the CodeLaunch deployment PDF: it explicitly notes that the GitHub repo (at the time of that writeup) is basically **two minimal Express services** (Orchestrator + Worker), and that it lacks the core architectural pieces proposed in the whitepaper (runtime/ECS, inference service, schema packages, etc.).

0 Ghost Engine - updated roadma...

That’s basically the “drift” in one sentence: *skeleton services exist; kernel doesn’t yet feel like the star.*

3) Marble research (the “why this feels like it might need a pivot” signal)

- Marble Worlds - ... (from Claude).md

This is a big deep dive on Marble’s role in pipelines. The key takeaway (for Ghost Engine positioning) is actually *very supportive* of your original thesis:

- Marble is framed as **not a complete game engine**.
- It’s primarily a **world/environment generator** whose outputs become truly interactive only after **export + import into an engine/runtime** where physics, object interactions, game logic, etc. are added.

Marble Worlds - From Virtual Ge...

- Marble AI - Grok convo1.pdf

Supplemental research about Marble developer resources/integration patterns.

Marble AI - Grok convo1

4) “Startup/pitch” support material

- 3 Additional source materials for ChatGPT.zip

Business/growth strategy books. Helpful for CodeLaunch storytelling and positioning, but not product/kernel implementation momentum.

Where the folder shows you’ve drifted from the OSS vision

Here’s the cleanest way to say it:

Drift 1: DevPost scoring incentives became an implicit “product spec”

The DevPost rules push “multiple Cloud Run services,” “AI tooling,” and “publish content” for points.

DevPost - Cloud Run Hackathon R...

That naturally leads to orchestrator/worker service design, deployment automation, WIF, etc.

That stuff is not “bad,” but it easily turns into the main thing—especially when the kernel doesn’t yet have a tangible, runnable reference implementation that everyone can point at.

Drift 2: The implementation center looks infra-first rather than kernel-first

The CodeLaunch deployment doc explicitly describes the repo as **two minimal Express microservices**, missing the larger kernel/runtime pieces. 0 Ghost Engine - updated roadma...

So the folder’s “mass” is: planning + deployment + contest compliance + docs, rather than “kernel + demo + plugin contracts.”

Drift 3: Marble can tempt a *product identity swap*

Your Marble deep-dive material is rich—transmedia pipelines, workflows, export formats, etc.—but it can pull you toward:

- “We are a content pipeline / world-gen platform wrapper”
instead of:
- “We are the deterministic interactive kernel + orchestration layer that makes world-gen outputs playable.”

The Marble report itself is clear that Marble outputs aren’t object-interactive “game worlds” by default; they need an engine/runtime layer for interactivity. Marble Worlds - From Virtual Ge...

That’s your opening, not your replacement.

The key insight: Marble *reinforces* Ghost Engine’s original thesis

If Marble (and “Large World Models”) are real, then the need for a **bridge layer** that:

- preserves deterministic state,
- adds interaction/physics/gameplay logic,
- and orchestrates generative modules safely

...gets *more* obvious, not less.

So the pivot probably isn't "pivot away from Ghost Engine OSS vision."

It's: re-center the OSS kernel, and use Marble as one demo-grade provider module.

A "slight pivot" that keeps you on-vision for CodeLaunch

If you want to acknowledge the Marble moment *without losing the plot*, pivot the CodeLaunch OSS submission story to:

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

Concretely:

1. OSS Core = deterministic kernel + plugin interfaces

- Make the public OSS artifact a reference kernel: `WorldState`, `Tick`, `Events`, minimal ECS/scene graph, etc.
- Everything "generative" is a module behind interfaces.

2. Marble becomes a `WorldProvider` plugin

- `WorldProvider` interface: `generate()`, `fetchAssets()`, `getColliders()`, `metadata()` etc.
- Provide two implementations:
 - `MarbleProvider` (real API)
 - `SampleProvider` (ships with pre-generated assets so anyone can run the OSS demo without Marble access)

3. Your demo becomes "Generate → Import → Interactive"

- Marble generates the environment.
- Ghost Engine runs authoritative state + interactions.
- Renderer: your Phase 2 3D + your Phase 1 ASCII as a "determinism debug view."

4. Cloud Run/AWS become optional deployment targets

- Keep them as adapters, not the identity.
- Contest-compliance scaffolding goes in `/infra/` or `/archive/`.

A quick "are we back on vision?" checklist

You're aligned with the OSS vision again when these are true:

- One **authoritative world state** exists, independent of AI output.
- The same world state can drive **ASCII + 3D** (engine-agnostic proof).
- Generative calls are **modules** that can be swapped/disabled.
- You store **provenance** (prompts, seeds, asset hashes).
- The demo runs **locally** without needing Cloud Run or cloud accounts.

What I'd archive vs. promote (based on this folder)

Promote (make central to CodeLaunch OSS narrative)

- The "kernel / imagination / orchestration" architecture doc and anything that defines contracts and core runtime scope. 0 Ghost Engine - updated roadma...
- Marble report sections that explicitly say "Marble isn't the engine; you need an engine/runtime for interactivity." That's basically your positioning proof. Marble Worlds - From Virtual Ge...

Archive (still useful, but not identity-defining)

- DevPost Cloud Run hackathon rules + Cloud Run scoring optimization notes. DevPost - Cloud Run Hackathon R...
- Most of the infra setup transcripts / contest checklists
- Polkadot contest exploration (unless it becomes a deliberate roadmap track)

If you want, I can also produce a "**vision** → **implementation diff**" for CodeLaunch: a concise list of the *exact kernel primitives* to implement first (so you stop paying "infra tax" before the kernel is real), plus how to frame Marble as a plugin demo rather than a pivot away.

Sources



since everything AI seems to be a constant moving target, what do you estimate Ghost Engine OSS' shelf life of relevancy being?

I'm ballparking its proposed needle skip will start to be moot by Nov 2026?

According to a document from 12/23/2025 ("updated roadmap for CodeLaunch"), Ghost Engine OSS' *center of gravity* is:

- a **deterministic simulation kernel** (world state + physics / ticking)
- an **imagination layer** (AI gen for images/video/audio)
- an **orchestration fabric** (scheduling modules, caching, fallback/policies)
- plus an optional **agent layer** (NL interface to orchestrate tools/build automation)

That's the "hybrid framework that plugs generative modules into a deterministic engine" idea.

What your project folder suggests you've drifted toward

From what's in this ChatGPT project + your Dropbox screenshot (Phase 1 ASCII builds, Phase 2 3D builds, demo runs, post-mortem bundle), the emphasis has increasingly become:

- **building a showcase game (GE DOOM)** and its renderer/runtime (ASCII → 3D),
- and **shipping infra/ops** (deployments, CI/CD, dashboards, etc.)

That's not "wrong" (CodeLaunch judges want a working demo), but it can quietly invert the OSS pitch:

- **OSS vision:** "we're an *integration kernel* that makes AI outputs reproducible + interactive across engines/providers"
- **where you've been heading:** "we're building *a game engine/game* (with AI somewhere in the loop)"

In other words: the reference experience (GE DOOM) started becoming *the product*, instead of a *proof harness* for the kernel.

Why Marble matters—and why it *doesn't* kill the kernel

Marble (World Labs) looks like it collapses a big chunk of the "Phase 2 3D" pain by giving you fast, photoreal world generation and exports (splats + meshes + colliders) designed to drop into Unity/Unreal/web.

But Marble also *explicitly* sits as a "generative booster" in the pipeline—not the interaction/simulation layer.

And the doc you uploaded notes current gaps that still require “engine/kernel” work: physics properties aren’t auto-generated, placement control is limited, meshes may need UV cleanup, etc.

So the smart framing isn’t “pivot away from Ghost Engine OSS to Marble.” It’s:

Make Marble your first marquee *adapter* inside Ghost Engine OSS.

That actually *strengthens* the OSS story: “we’re vendor-agnostic; Marble is one provider; tomorrow it’s others.”

What I’d pivot for the CodeLaunch OSS submission (without throwing away Phase 2)

If you want the submission to re-align with OSS vision *and* exploit Marble:

1) Recast GE DOOM as a “reference runtime,” not the product

GE DOOM becomes the demo harness that proves:

- deterministic tick / replay
- “world ingestion” from a generator (Marble today; others later)
- conversion into an interactive scene (colliders → navmesh → spawn points → pickups → triggers)
- caching + provenance (“this build used world hash X, prompt Y, seed Z”)

2) Narrow Phase 2 “3D” to *interaction*, not rendering

Let Marble/engine plugins handle rendering quality; Ghost Engine handles:

- **scene decomposition** into gameplay-relevant layers (collision, interactables, spawn graph)
- deterministic simulation + net-safe state updates
- policy/caching/versioning
- adapters (Marble, local assets, other gen tools)

This also future-proofs you if Marble improves output fidelity; your kernel remains the glue.

3) Build the CodeLaunch demo around “AI→Interactive in minutes”

A tight demo loop judges can grok:

1. Submit prompt (or pick a preset)
2. Orchestrator calls Marble → gets world export
3. Kernel ingests → builds collider/trigger map → spawns DOOM loop
4. Player runs the level

5. Re-run with same seed = same layout (determinism); new seed = new level

You're no longer trying to out-engine Unity/Unreal—your differentiator is **reproducible AI-to-gameplay**.

Shelf life: will the “needle skip” pitch be moot by Nov 2026?

Your ballpark is pretty reasonable if the pitch is **ONLY** “we bridge AI gen into game engines.”

Marble's own “expected enhancements” list (in the doc you uploaded) includes things that would directly compress your novelty window: world composition, physics parameters, and eventually interactive element APIs.

And the longer-term vision described is basically “fully decomposable worlds” with conversational scene editing and layered affordances.

My practical estimate

- If Ghost Engine OSS = “needle skip” only: differentiation starts fading in 12–18 months; by late 2026 it's likely not special.
- If Ghost Engine OSS = “open orchestration + determinism + provenance + eval + adapters”: shelf life is **multi-year**, because even if generators add “interaction,” teams still need:
 - reproducibility across vendors/versions
 - caching/cost controls
 - audit/provenance
 - portability across engines
 - regression/eval harnesses (“did the new model break collision? nav? gameplay metrics?”)

That second framing is how you make “moving target AI” work *for* you: every new model/API becomes “just another adapter,” and your kernel becomes the stable layer.

If you want, I can turn this into a **1-page CodeLaunch positioning statement** (what OSS is, what GE DOOM is, and where Marble plugs in) so the repo + pitch deck stop fighting each other.

Sources



