Part III — Pipelines & Workflows

The data pipeline

The path every piece of battlefield information takes:

```
map files (facts)  ⟶  rules layer (judgments)  ⟶  consumers (pathfinding, combat, AI, rendering)
```

1. **Maps are facts.** A `.vmap` file (or a generator) holds voxels: material, shape, rotation, height.
2. **The registry names the materials** (append-only). The codec checks catalog versions so old maps can't silently load wrong.
3. **The rules layer interprets.** Consumers never read "walkable" from the map — they read facts and ask the rules.
4. **Consumers decide.** Pathfinding, and later combat and AI, produce verdicts (reachable, cost, damage) that are *reported*, never written back into the map.

The build & test pipeline

The guardrail that keeps the game trustworthy:

1. **`./run_tests.sh`** imports the project (so new classes register), then runs the whole headless test suite.
2. The suite covers the codec round-trips (byte-exact), the append-only catalog contract, and every subsystem's core behavior — including all of the pathfinding pipeline.
3. A **pre-commit hook** runs the suite; a commit that turns the suite red is rejected.
4. Tests are small, focused files under `tests/`, auto-discovered by name.

Rule: no red commits, ever. When behavior changes, the tests change with it.

The content & asset pipeline

How maps and content get made:

- **Builders** (`tools/*.gd`, run from Godot) construct demo maps, the test "museum" map, and the voxel demos programmatically, then pack them through the same codec any game would use.

- **The test map** is the scenario museum — every hazard, elevation shape, layered structure, and wall the model can express — and its tests assert the map's shape so builders can't drift.
- Future **procedural generation** and the **in-game editor** are just other producers of the same map data, flowing through the same codec and renderer.

The docs & wiki pipeline

The documentation is versioned in the repo and mirrored to the wiki:

1. **Canonical docs live in the repo:** this bible, `MAP_RULES.md`, `CONTEXT.md`.
2. **`tools/sync_wiki.py`** reads a manifest (`tools/wiki_manifest.json`) and pushes each doc (and each bible part) to a BookStack book as chapters and pages — creating or updating in place.
3. The wiki is a **rendered view**; the repo is the source of truth. When the game changes, the docs change first (rule below) and the sync re-mirrors them.

The ship workflow

- Branch `main`; commits are concise, lowercase, imperative, focused on "why"; push frequently to the Forgejo remote.
- Repo uses the **SHA-256 object format** — never recreate it as sha1, never rely on push-to-create.
- When the game is ready to distribute on Arch Linux, a PKGBUILD makes it installable with `paru` (see `CONTEXT.md`).

Revision #10
Created 2026-08-15 18:17:02 UTC by Fifthdread
Updated 2026-08-15 19:00:00 UTC by Fifthdread