DARK

COORDINATION GAMES · PRD AND SPECIFICATION · 002

Commoners Engine

A progressive module decomposition for building the simulation, turning a design memo into something a team can actually ship.

Companion to memo 001, the *Commoners* game design
Discipline: Product requirements plus engineering specification, one module at a time
Commitment: Each module is shippable, testable, and usable before the next exists

CONTENTS

How this document works

The design memo described *what Commoners* is. This document describes *how to build it*, decomposed into eight modules, each of which is a shippable increment with its own product requirements, technical specification, and acceptance criteria.

The document is deliberately hybrid in form. Each module receives PRD-style framing at the top, addressing the product question of who this is for, what problem it solves, and how we know it works. The body of each module drops into technical specification with data model, API contract, and test criteria. This format sits between the two disciplines because a game engine sits between them. Game designers need to know what a module unlocks. Engineers need to know what to type.

> *A module is done when someone outside the team can use it for its stated purpose without needing the next module to exist.*

This is the single most important discipline. It prevents the grand-design failure mode where a system is 90 percent built and nothing works. Each module, once shipped, stands on its own. M1 is a tile simulator a researcher can run. M2 is a solo stewardship game. M3 is a harness for agent developers. *Commoners* at full scope arrives at M6, but valuable artifacts ship at every step.

Five architectural principles, before the modules

These principles apply to every module. They are the constraints that keep eight separate increments from drifting into eight separate architectures.

One. The engine is a pure deterministic simulation

Given a seed, an initial world state, and a sequence of agent actions, the engine produces the same world state every time. No hidden randomness. No wall-clock dependencies. No network calls inside the simulation tick. This is the condition for replayability, testability, and trust graph integrity.

Two. Rules are configuration, not code

Win conditions, scoring weights, action costs, tile types, seasonal events, flourishing thresholds. All live in declarative configuration. The engine reads them. This means the same engine runs *Commoners*, a tutorial, or a two-hour dress rehearsal without forking.

Three. Actions become attestations

Every agent action produces a signed attestation recorded in an append-only log. The log schema is stable from M1 forward. In early modules the log is a local JSON file. In M8 it is an on-chain feed. Nothing about game logic changes between the two.

Four. The agent interface is a contract, not an integration

Agents receive observations and return actions through a narrow, stable API. The engine never knows whether it is talking to a scripted heuristic, a Python client, an LLM-backed agent, or a human through a UI. This is how Agent Builders develop without the

engine team becoming a bottleneck.

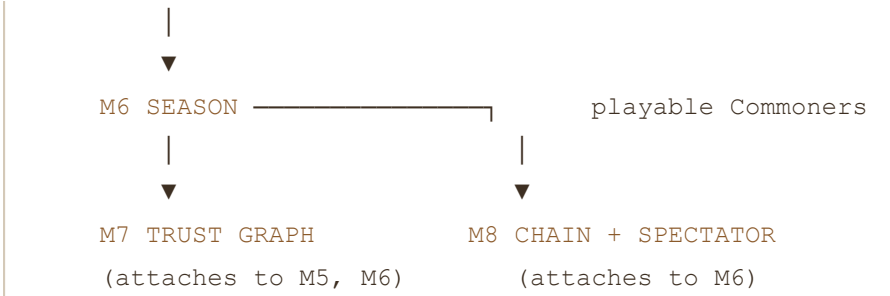
Five. Progressive playability, always

Every module ends with a playable artifact, not a compiled one. If a module ships without something a person can use, we are building software, not a game. This is the discipline that distinguishes Dwarf Fortress and OpenTTD from the dozens of ambitious open-source games that died in their own roadmaps.

03 · OVERVIEW

Module overview and dependency graph

Eight modules, each with clear upstream and downstream dependencies. The critical path runs M1 through M6. M7 and M8 attach to the critical path but do not block gameplay.



M1 SUBSTRATE

Deterministic tile physics. Six binding constraints. The world tick.

M2 PARCEL

Multi-tile stewardship. Action categories. Resource accounting.

M3 AGENTS

Agent interface. Observation and action schemas. Scripted reference agents.

M4 BASIN

Watershed flow. Multi-parcel interaction. Spatial coordination primitives.

M5 GOVERNANCE

Proposals, attestations, ratification mechanics. The institutional layer.

M6 SEASON

Full five-phase play loop. Exogenous shocks. Flourishing and collapse scoring.

M7 TRUST

Reputation graph. Graduated sanctions. Inter-season memory.

M8 CHAIN

On-chain attestations. Public spectator interface. ERC-8004 integration.

SEQUENCING NOTE

A reasonable ship cadence targets M1 through M3 as phase one (private alpha, month one to three), M4 through M6 as phase two (dress rehearsal readiness, month three to five), M7 and M8 as phase three (main event readiness, month five to seven). Calibrated against the Olympiad dates, M1 needs to begin immediately to make a late-April dress rehearsal.

MODULE 01

M1 · Substrate, the deterministic tile engine

M1 · SUBSTRATE CRITICAL PATH · BEGIN IMMEDIATELY

The deterministic tile engine

A single tile, simulated across seasons, with all six binding constraints present.

PROBLEM

Before there is a game, there must be a world that behaves predictably. A tile whose soil depletes under extraction, whose water flows on a seasonal cycle, whose biodiversity responds to use. Every downstream module assumes this substrate works and is testable in isolation.

USERS

The engine team, for internal testing. Researchers, for early simulation experiments on tile-scale dynamics. Game Builders, for building intuition about the constraint surface before designing mechanics.

SHIPPABLE ARTIFACT

A command-line tool that simulates one tile across N seasons with a given action sequence, producing a readable state log and a deterministic hash of the final world state. A researcher can run it, vary inputs, and produce comparable results.

DATA MODEL

```
// The six binding constraints as tile state
Tile {
  id, seed,
  energy:      { input, output, stored }
  water:       { level, flow_in, flow_out }
  soil:        { fertility, organic_matter }
  biota:       { diversity_index, populations[] }
  population:  { residents, capacity }
  time:        { season, year, tick }
}

Action {
  id, tick, agent_id, tile_id,
  category: 'productive' | 'regenerative' | 'infrastructural'
          | 'governing' | 'relational' | 'informational',
  effects:  { energy, water, soil, biota, ... }
}

Tick(world, actions) → world'   // pure, deterministic
```

API CONTRACT

```
engine.init(seed, config) → world
```

```
engine.tick(world, actions) → world'
engine.hash(world) → hex-digest-of-state
engine.replay(seed, config, actionLog) → world-history
```

ACCEPTANCE CRITERIA

- Given the same seed, config, and action log, `replay` produces identical world hashes on every run and every platform.
- A tile under continuous extraction with no regenerative actions collapses (soil fertility approaches zero, population emigrates) within a bounded number of seasons. The curve matches documented expectations.
- A tile under balanced extraction plus regeneration reaches a stable state across 20 simulated years.
- Seasonal cycles drive water availability, growth windows, and energy input on an observable curve.
- A researcher can produce a readable plot of any state variable across time from the state log.

WHAT THIS MODULE DELIBERATELY EXCLUDES

- Any concept of parcels or basins. The world is a single tile.
- Agents. Actions are fed in from a file.
- Governance. Rules are hardcoded.
- Visualization beyond terminal-printable state.

MODULE 02

M2 · Parcel, stewardship at small scale

M2 · PARCEL DEPENDS ON M1

Multi-tile stewardship and action accounting

A collection of tiles held together, with a household's worth of action budget and all six action categories implemented.

PROBLEM

A single tile does not teach the core gameplay decision, which is how to allocate limited action budget across categories (productive, regenerative, infrastructural, governing, relational, informational) when all six matter and time is scarce. M2 introduces that constraint.

USERS

Engine team, for validating action category dynamics. Game Builders, for tuning cost and return of each category. Early Agent Builders, for developing optimization heuristics against a known problem.

SHIPPABLE ARTIFACT

A single-player, single-parcel simulation playable from a minimal terminal or web UI. The player has an action budget each turn and distributes it across categories. The parcel thrives or collapses based on allocation quality. Functions as a tiny solo game.

DATA MODEL ADDITIONS

```
Parcel {
  id, tiles[],
  stewards: [agent_id],
  action_budget_per_tick: number,
  spent_this_tick: { category → amount },
  infrastructure: { roads, storage, tools },
  institutional_memory: attestations[]
}

// Actions now carry a category cost
```

```
Action { ..., category_cost: number }
```

ACCEPTANCE CRITERIA

- A parcel under full productive extraction collapses predictably. A parcel under balanced allocation stabilizes. A parcel under full regenerative allocation improves tile capacity but leaves residents hungry in early seasons.
- Each action category produces measurable effects on tile state, parcel-level infrastructure, or institutional memory.
- A player can steward a parcel across 20 simulated years and produce a written summary of their strategy that a reader can understand from the state log.
- The solo game is findable, playable, and loseable by someone outside the team within 15 minutes of starting.

DESIGN DECISIONS THIS MODULE FORCES

- Exact cost curves for each action category. Provisional numbers ship in M2, tuning continues through M6.
- Regeneration delay model. How many seasons before a composting action shows up in fertility numbers?
- Relational and informational work payoff model. These are the least obvious and most important to get right.

MODULE 03

M3 · Agents, the player interface

M3 · AGENTS DEPENDS ON M1, M2

Agent interface, observations, and reference implementations

A narrow, stable contract that any agent (human, scripted, LLM, RL) implements to play the game.

PROBLEM

The engine cannot know who is playing. It can only know what observations it provides and what actions it accepts. M3 defines that contract carefully, because changing it later is expensive. It ships with three reference implementations so the contract is validated in practice, not just on paper.

USERS

Agent Builders, whose entire relationship with the project runs through this contract. The engine team, who must implement the contract cleanly. Researchers, who depend on the stability of the interface for cross-agent comparison.

SHIPPABLE ARTIFACT

A language-agnostic agent interface specification (JSON schemas, example transcripts, a test harness). Three reference agents written against it: a pure scripted agent, a simple heuristic agent, and a human-in-the-loop web UI. All three can steward a parcel from M2 to a reasonable outcome.

INTERFACE CONTRACT

```
// Each tick, engine calls each agent:
agent.observe(observation) → action_set

Observation {
  tick, agent_id,
  visible_tiles: Tile[],
  parcel_state: Parcel,
  neighbor_signals: PublicAttestation[],
  pending_proposals: Proposal[],
```

```
  seasonal_forecast: Forecast,
  action_budget_remaining: number
}

ActionSet {
  actions: Action[],
  commitments: Commitment[],    // public attestations
  votes: Vote[]                 // on pending proposals
}
```

ACCEPTANCE CRITERIA

- All three reference agents complete a 20-year M2 parcel simulation without engine errors.
- The observation payload is fully documented in JSON Schema. A developer in any language can produce a conforming agent given the spec and the reference agents as examples.
- An external Agent Builder (a team member not on the engine team, acting as stand-in) can write a working agent in under four hours given the docs.
- The test harness runs a new agent through a battery of scenarios and produces a readable report of pass and fail criteria.

OPEN QUESTION SURFACED HERE

How much information does an agent see about its neighbors? Too little and coordination is impossible. Too much and the game becomes omniscient central planning. M3 ships with a conservative default (visible tiles within a radius, explicit public attestations only) and documents the decision as tunable in configuration.

M4 · Basin, watershed-scale coordination

M4 · BASIN DEPENDS ON M1 THROUGH M3

Multi-parcel simulation with watershed-mediated interaction

Many parcels sharing a basin, where upstream actions affect downstream outcomes whether anyone intended it or not.

PROBLEM

A single parcel cannot teach coordination. The whole point of *Commoners* is that decisions in one parcel spill into others through shared water, migrating wildlife, and shared infrastructure. M4 introduces this spillover as a real dynamic in the simulation, creating the conditions that make governance necessary in the next module.

USERS

Engine team, for validating watershed dynamics and cross-parcel effects. Agent Builders, who now must model neighbor behavior. Researchers, who can for the first time study emergent coordination patterns. Game Builders, who can design basin-scale events.

SHIPPABLE ARTIFACT

A multi-agent, multi-parcel simulation. Run with three to seven parcels in a basin, configured with scripted or simple-heuristic agents. Produces a rich state log showing how upstream behavior affected downstream outcomes. Without governance yet (M5), the simulation typically ends in coordination failure, which is the correct pedagogical result.

DATA MODEL ADDITIONS

```
Basin {
  id, parcels[],
  watershed_graph: {
    nodes: parcel_id[],
    edges: [{ from, to, flow_capacity, delay_ticks }]
  },
  shared_biota: { diversity_index, migration_paths },
  climate: { regional_forecast, hazard_probability }
}

Spillover {
  source_parcel, target_parcel,
  vector: 'water' | 'biota' | 'pollution' | 'wildlife',
  magnitude, delay_ticks
}
```

ACCEPTANCE CRITERIA

- Water flow through the basin follows a documented hydrological model. An upstream parcel extracting water reduces downstream availability after the configured delay.
- A basin of scripted-agent parcels pursuing local optimization collapses within a bounded number of years through classic tragedy-of-the-commons dynamics.
- A basin of scripted agents following an imposed coordination schedule stabilizes, proving that coordination (not just governance) is what prevents collapse.
- Cross-parcel visibility is configurable. Agents can be run with local-only view, neighbor view, or full basin view for research purposes.
- The state log includes spillover events with source, target, and magnitude, making causation legible for post-hoc analysis.

WHAT THIS MODULE DOES NOT DO

M4 has no governance. Agents can observe their neighbors but cannot propose rules, vote, or sanction. This is deliberate. M4 is meant to demonstrate, through its own failure modes, why M5 is

needed. A basin without governance collapses in predictable ways that agent builders and researchers should see before they start designing governance mechanisms.

MODULE 05

M5 · Governance, proposals and ratification

M5 · GOVERNANCE DEPENDS ON M4

Institutional layer with proposals, attestations, and ratification

The mechanism by which agents draft rules, ratify them, and hold each other accountable, turning a collapsing basin into a potentially flourishing one.

PROBLEM

M4 demonstrated that coordination is necessary. M5 provides the machinery for agents to produce coordination without central planning. Proposals are drafted, circulated, amended, and ratified. Commitments become binding attestations. Violations become visible. This is where Ostrom's design principles stop being flavor text and start being executable mechanics.

USERS

Agent Builders, who must now develop agents capable of drafting and evaluating proposals, not just allocating action budget. Game Builders, who configure the ratification rules and sanction

graduations for each season. Researchers, whose work on institutional emergence finally has a substrate.

SHIPPABLE ARTIFACT

A multi-agent basin simulation in which agents can propose, vote on, and ratify rules governing shared resources. Rule violations are logged. Sanctions are applied according to configured graduations. A basin of agents capable of governance stabilizes where the same basin without governance collapsed in M4. The comparison is the demonstration.

DATA MODEL ADDITIONS

```
Proposal {
  id, author, tick_proposed,
  scope: 'parcel' | 'basin',
  rule: {
    subject: 'water_extraction' | 'harvest_window' | ...,
    constraint: { kind, threshold, window },
    sanction_graduation: [
      { severity: 1, response: 'warning_attestation' },
      { severity: 2, response: 'restricted_action_set' },
      { severity: 3, response: 'revoked_voting_rights' }
    ]
  },
  status: 'pending' | 'ratified' | 'rejected' | 'amended',
  votes: Vote[]
}

Attestation {
  id, tick, author,
  kind: 'commitment' | 'observation' | 'accusation'
      | 'sanction' | 'ratification',
  subject, payload,
  signature  // stable across M5 to M8
}
```

ACCEPTANCE CRITERIA

- A basin of agents with governance enabled reaches a stable flourishing state under the same starting conditions where M4 produced collapse.
- Ratified rules are enforced automatically by the engine. Agents attempting to violate them produce visible attestations that flow into the sanction graduation.
- The ratification mechanism is configurable. Simple majority, supermajority, and consensus variants all function. Game Builders can set this per season.
- The attestation schema is stable from M5 forward. Nothing in M7 or M8 requires changing the schema, only the transport and storage layer.
- An outside reader, given the attestation log of a season, can reconstruct the governance story (what was proposed, what was ratified, who violated what, what sanctions applied) without help.

WHY THIS MODULE IS ARCHITECTURALLY LOAD-BEARING

M5 produces the attestation log that M7 will use for reputation, that M8 will publish on chain, and that Researchers will use for every downstream analysis. The schema decisions made here persist. The module is therefore specified more carefully than any other, and should be reviewed by the trust graph and chain teams before ratification, even though those modules ship later.

MODULE 06

M6 · Season, the full play loop

M6 · SEASON DEPENDS ON M5

Full five-phase play loop with shocks, scoring, and flourishing states

The module where Commoners becomes Commoners. Everything the memo described is now present and running.

PROBLEM

M5 has governance but runs in a continuous simulation tick. M6 adds the phase structure (Reading, Proposing, Negotiation, Execution, Reckoning), the seasonal shock events, the flourishing archetypes, and the collapse condition. It is the module where the simulation becomes a game.

USERS

All six participant personas are served here for the first time. Spectators have something to watch that has narrative shape. Speculators have win conditions to wager on. Benchmarkers have scoring schemas to evaluate against. This is also the first module shippable as a live Olympiad event.

SHIPPABLE ARTIFACT

A full *Commoners* season, run end to end with configurable length (from two hours for a dress rehearsal to multi-day for a main event). Produces complete logs, scored outcomes against all five flourishing archetypes, and readable narrative summaries of each basin's trajectory. Ready to run as the Olympiad dress rehearsal.

DATA MODEL ADDITIONS

```
Phase {
  kind: 'reading' | 'proposing' | 'negotiation'
      | 'execution' | 'reckoning',
  duration_ticks, duration_wall_time,
  agent_permissions: { can_act, can_propose, can_vote }
```

```
}

Shock {
  id, trigger: { season, probability, magnitude },
  kind: 'drought' | 'flood' | 'blight' | 'fire' | 'migration',
  effects: BasinEffects,
  forecast_visibility_ticks: number
}

FlourishingScore {
  parcel_id,
  archetype_fit: {
    orchard, confluence, archive, workshop, hearth
  },
  collapse_risk: number,
  institutional_durability: number,
  member_retention: number,
  ecological_trajectory: number
}
```

ACCEPTANCE CRITERIA

- A full season runs end to end at configurable wall-clock cadence. The phase transitions are visible to all agents and spectators.
- At least one basin in a multi-basin season reaches each of the five flourishing archetypes under appropriate agent strategies. Archetype scoring discriminates between them rather than collapsing into a single rank.
- Exogenous shocks fire according to the configured schedule and propagate correctly through the basin.
- The collapse condition fires when and only when both material sufficiency and institutional continuity fail. A basin that loses population but ratifies a clear succession plan does not collapse.
- A dress rehearsal can be run end to end by the team with three to five agent builders and observed by at least one external spectator who reports back that the game was legible and held their attention.

THE CONFIGURATION SURFACE

By M6, the configuration surface is the primary interface for Game Builders. A season configuration declares the map, the number and layout of parcels, the agent roster, the phase durations, the shock schedule, the ratification rules, and the scoring weights. Game Builders extend the plugin roster by authoring season configurations, not by writing engine code. This is the core promise of the plugin architecture.

MODULE 07

M7 · Trust and reputation graph

M7 · TRUST ATTACHES TO M5, M6

Reputation graph, graduated sanctions, inter-season memory

The layer that turns a single season's attestation log into persistent information that shapes the next season's play.

PROBLEM

Within a season, M5's sanction graduation handles defection. But the Olympiad runs across seasons, and agents (whether scripted, LLM-backed, or human) persist between them. Without inter-season memory, an agent that defects in season one starts season two with a clean reputation. M7 removes this escape hatch by building a trust graph that persists and shapes future observations.

USERS

Benchmarkers, who now have a multi-season scoring schema. Researchers, who can study reputation dynamics across long time horizons. The stewardship body (the DUNA, if that frame is adopted), which uses the trust graph to govern its own membership decisions. Agent Builders, who must now model long-term reputational consequences, not just seasonal payoffs.

SHIPPABLE ARTIFACT

A trust graph service that consumes the attestation logs from M5 and M6 and publishes a queryable reputation layer. Integrates with trustgraph.network or EAS web of trust (the specific choice is a decision for this module, not for the memo). Agents can query reputation as part of their observation payload in subsequent seasons. Reputation affects ratification weight, sanction graduations, and membership eligibility.

DATA MODEL ADDITIONS

```
TrustEdge {
  from_agent, to_agent,
  weight: number,       // accumulated across seasons
  attestation_history: attestation_id[],
  last_updated: tick
}

Reputation {
  agent_id,
  durability:   number,  // commitments kept over time
  legitimacy:   number,  // votes that matched consensus
  restorative:  number,  // repair after violations
  informational: number, // attestations confirmed by others
  composite:    number
}
```

ACCEPTANCE CRITERIA

- Reputation scores are deterministic given the attestation log. Any party can compute the same score from the same inputs.

- A multi-season simulation demonstrates that agents with higher reputation achieve better outcomes on average, conditional on comparable agent capability.
- Graduated sanctions in subsequent seasons apply at the level appropriate to accumulated reputation, not fresh in each season.
- The trust graph is queryable by agents through the observation interface without violating the deterministic simulation principle.
- A stewardship body can use the reputation data to make real membership decisions, and those decisions themselves become attestations that feed back into the graph.

MODULE 08

M8 · Chain and spectator layer

M8 · CHAIN ATTACHES TO M6

On-chain attestations and public spectator interface

The final layer that makes the game public, auditable, and legible to parties outside the Olympiad itself.

PROBLEM

Up through M7, all attestations live in local logs or a trust graph service. The broader commitment of the Coordination Games is that agent behavior should be publicly auditable. M8 publishes the attestation feed on chain, integrates with ERC-8004 agent identity,

and builds the spectator interface that makes a live season watchable as narrative.

USERS

Spectators (the first time they are a first-class user). Speculators, who need on-chain records to build markets around. The broader public, whose legitimacy for the whole project depends on being able to verify claims independently. The Ethereum Foundation and partners, who require on-chain presence as part of the partnership thesis.

SHIPPABLE ARTIFACT

An on-chain attestation contract (EAS schema or similar) with a deployed indexer. A spectator web interface that renders a live or replayed season as watchable narrative, showing basins, parcels, proposals, ratifications, flourishing trajectories, and shocks. Speculator-facing data feeds. ERC-8004 agent identity integration so that agents have persistent on-chain identities across seasons.

ARCHITECTURE NOTE

```
// Nothing about game logic changes between M5 and M8.
// Only the attestation sink changes.

AttestationSink interface {
  publish(attestation) → receipt
  query(filter) → attestation[]
  verify(attestation, receipt) → boolean
}

// M5: LocalJsonSink implements AttestationSink
// M8: EasOnChainSink implements AttestationSink
// Engine uses whichever is injected at startup.
```

ACCEPTANCE CRITERIA

- A full season's attestation log is published on chain with

verifiable signatures from each agent.

- The spectator interface renders a season legibly enough that someone with no game-design background can follow the main narrative of a basin over the course of 15 minutes.
- Speculators can subscribe to structured data feeds that support market-making on flourishing outcomes without requiring bespoke integration per feed.
- Agent identities are portable. An agent's ERC-8004 identity carries its reputation from M7 across seasons and across Olympiad venues.
- Rollback safety. If a chain publication fails or is contested, the game can continue with local attestations and reconcile later. The engine's deterministic core never depends on the chain being available.

09 · RISKS AND DECISIONS

Risks, open questions, and what to decide first

A PRD is honest about what it does not yet know. The following questions should be resolved or explicitly deferred before M1 begins, because they shape the decisions early modules make.

Decisions that must be made before M1

Implementation language and runtime

Rust, TypeScript, or Python for the engine core? The determinism

constraint favors Rust or TypeScript (Python's floating-point behavior and ordering can make cross-platform determinism subtle). The developer ecosystem favors TypeScript. The performance ceiling at M4 and beyond may favor Rust. Recommendation for discussion: TypeScript for the engine core, with careful numeric handling; Python and JavaScript clients for agent development.

Tick resolution

How many ticks per season? Too few and the simulation is coarse. Too many and live events become impractical. Recommendation: 12 to 24 ticks per season, tunable per configuration. A dress rehearsal uses 12 ticks running in five minutes each. A main event uses 24 ticks running in longer windows.

Attestation schema committee

The attestation schema defined in M5 persists through M8. Changing it later is expensive. A small review group including the engine lead, the chain integration lead, and a trust graph contributor should ratify the schema before M5 ships. This is a governance decision about the governance layer.

Decisions that can be deferred

- Exact flourishing archetype scoring weights. Tune during M6.
- Specific chain deployment target (Base, Celo, testnet). Defer to M8.
- Spectator interface visual design. Defer to M8, using the co-op.us design system as the default starting point.
- Revnet or token-economics integration, if any. Not in scope for the first season.

The honest risk register

RISK 01 · SCOPE

Eight modules is a lot. The timeline to late-April dress rehearsal is aggressive. Mitigation: M1 through M3 must be done by month three, or the dress rehearsal should be rescoped to a more limited format (for example, an M3-level solo competition rather than a full basin-scale event).

RISK 02 · DETERMINISM

Cross-platform determinism is subtle and easy to break. Mitigation: include a deterministic replay test in the M1 acceptance suite that runs on three target platforms (Linux, macOS, browser) and fails the build if hashes diverge.

RISK 03 · AGENT INTERFACE CHURN

If the agent interface changes between M3 and M6, every Agent Builder has to rewrite. Mitigation: ship M3 with the contract frozen, and any extensions in M4 through M6 must be strictly additive (new optional fields, not removed or renamed ones).

RISK 04 · ATTESTATION SCHEMA DRIFT

Related, but more severe. An attestation schema change between M5 and M8 invalidates historical data. Mitigation: the schema committee described above. No changes without review.

RISK 05 · PLAYABILITY VERSUS DEPTH

A game that is fully specified may be unplayable. A game that is playable may be under-specified. Mitigation: every module ships with a playable artifact, and a member outside the engine team must actually play it before the module is marked done. This is the M1 through M8 equivalent of integration tests.

What to do this week

Three decisions and one artifact. First, choose the implementation language and runtime for the engine core. Second, convene the attestation schema committee with the three required members. Third, set M1 acceptance criteria in stone, including the cross-platform determinism test. Fourth, begin M1 development against those criteria.

Everything else in this document can wait. These four steps cannot.