

# Game-Creation-Agents: Autonomous Game Production

Multimodal Reinforcement Learning Team, Tencent

 <https://github.com/Tencent/Game-Creation-Agents>

**Abstract.** Game-Creation-Agents (GCA) turns a natural-language request into a runnable Phaser 4 browser game. The pipeline first builds a multimodal game design document that couples mechanics and scene contracts to grounded visual references. It then projects that document into a runtime scaffold, machine-readable contracts, and deterministic tests. Eight specialized agents next carry out production: scenes are implemented serially, assets are produced in parallel, and one reduced manifest joins the two branches into a single build. Build-linked real-UI checks and a code-blind critic then verify that build and decide whether the game may ship. Each blocker returns to the agent that owns the affected artifact for a bounded repair, and the game is released once all the gates pass. We evaluate finished games separately, through their rendered interfaces alone, over prompt-matched tasks run under a common protocol. GCA achieves stronger requirement satisfaction, core mechanics, content depth, and functional visuals than a comparable open pipeline. In a side-by-side human study, raters also score GCA’s games higher on every dimension and prefer them overall.

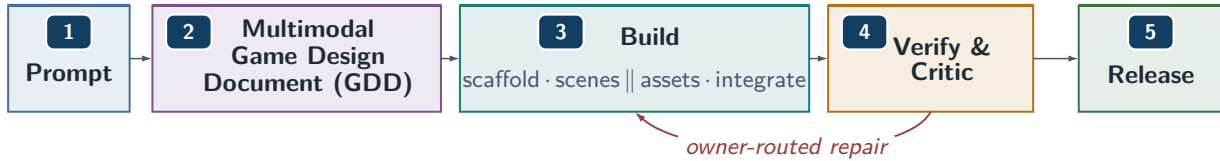


Figure 1: GCA pipeline. A prompt becomes a design document, then a build that must pass verification before release; blockers route back to their owners.

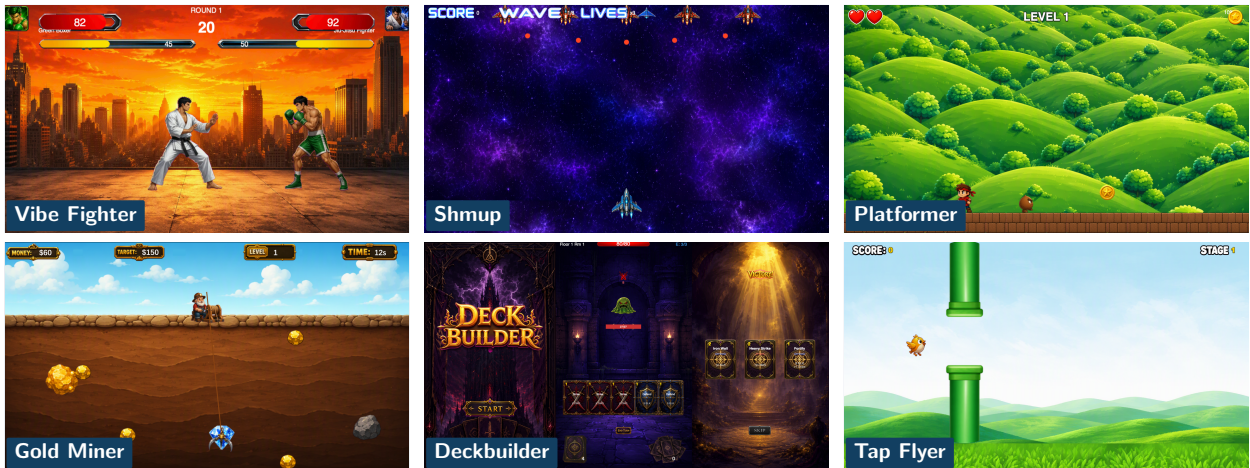


Figure 2: Six example games generated by GCA, spanning distinct mechanics and visual styles.

## 1 Introduction

Large language model agents can now write and revise real programs, and recent work organizes them into pipelines where each agent takes on a role (Qian et al., 2024; Hong et al., 2024; Wu et al., 2024). Games are a good target for this, and a hard one. A game is not finished when it compiles and renders. It has to take input, follow its own rules, move through levels and end states, and show the player what is going on, using art that was generated along with the code. One request therefore asks for dozens of things that can be checked, spread across the design, the running code, and the assets.

The hard part is that nothing outside the pipeline tells it when it is done. Code generation usually runs against a test suite that already exists (Chen et al., 2021; Austin et al., 2021; Jimenez et al., 2024), but a game pipeline has to build its own contracts, tests, and runtime evidence before it can judge what it made. This is the test-oracle problem (Barr et al., 2015). An agent reporting that a scene works is not evidence that it works (Huang et al., 2024), and a page that renders can still have broken input, collisions, progression, end states, or interface feedback. Running work in parallel makes it worse: design choices constrain scenes and assets, code and art are produced at the same time but have to match at integration, a repair can touch a file another agent owns, and leftovers from an earlier attempt can look like current output. Without a record of which artifact belongs to which agent and to which build, it is easy to mistake work being done for work being finished.

GCA takes a game request in plain language and produces a runnable Phaser 4 browser game (Phaser Studio, 2025), along with its design package, generated assets, executable tests, and verification evidence. Eight agents split the work, each owning one kind of artifact. The design agent writes a design specification paired with grounded scene concepts, detections, and element crops; together these are the *multimodal game design document (GDD)*, or *multimodal GDD*, and everything downstream is built from it. Scenes are then implemented one at a time, while independent asset batches run in parallel. Figure 1 gives the whole sequence, and Figure 2 shows six games it produced.

Verification then runs on the assembled game, not on what the agents said about it, and every check is bound to the build it ran against. Because each artifact has exactly one writer, every blocker has a responsible owner, and the repair goes back to that agent rather than to whoever noticed it. A typed Model Context Protocol (MCP) control plane (Anthropic, 2024) enforces both of these rather than leaving them to the agents.

We evaluate the finished games from outside the pipeline. The evaluator gets only the prompt and one finished game through its rendered interface. It never sees the multimodal GDD, the trace, the reports, or the critic’s verdict, so the internal gates cannot affect the score. Over prompt-matched tasks scored under one protocol, GCA has higher mean Requirement Satisfaction, Core Mechanics, Content Depth, and Functional Visuals than an open-source baseline (Jiang et al., 2026b), and performs comparably on Art & Presentation. A human study of the same pairs gives GCA higher mean ratings on all five dimensions and a clear majority of decisive preferences.

The rest of the report is organized as follows. Section 2 places GCA among prior work on complete-game generation, evaluation and testing, and content generation. Section 3 details the pipeline and Section 4 the MCP control plane. Section 5 reports the automated and human evaluations together with the Gold Miner case, and Section 6 summarizes the findings.

## 2 Related Work

We position GCA by distinguishing the artifact each system produces and the role assigned to execution evidence. Table 1 compares representative systems besides GCA through primary scope, evidence, and outcomes. GCA introduces composite evidence gates that rise from a seven-check scaffold boundary to the 39 checks that must all pass before the final game ships.

| System                               | Primary scope                             | Executed or rendered evidence                                              | Role of the outcome                                               |
|--------------------------------------|-------------------------------------------|----------------------------------------------------------------------------|-------------------------------------------------------------------|
| GameGPT (Chen et al., 2023)          | Multi-stage game development              | Task execution plus role-specific review                                   | Workflow review and completion                                    |
| OpenGame (Jiang et al., 2026b)       | End-to-end Phaser game and asset creation | Build/test verification; companion benchmark executes the game             | Iterative debugging; separately reported benchmark scores         |
| MarioGPT (Sudhakaran et al., 2023)   | Text-conditioned Mario-level generation   | Generated-level sampling and novelty search                                | Content generation and diversity                                  |
| GameDevBench (Chi et al., 2026)      | Godot development tasks                   | Engine validation; screenshots and runtime video studied as agent feedback | Development-agent benchmark                                       |
| OpenGame-Bench (Jiang et al., 2026b) | Finished web games                        | Headless browser execution and multimodal judging                          | Build Health, Visual Usability, and Intent Alignment scores       |
| GameCraft-Bench (Luo et al., 2026)   | Complete Godot games and replay traces    | Replayed inputs, gameplay video and frames, hidden rubric                  | Core Mechanics, Content Depth, Functional Visuals, and Art scores |
| GBQA (Jiang et al., 2026a)           | Bug discovery in generated games          | Interactive exploration and structured bug reports                         | QA-agent benchmark against verified bugs                          |
| Wuji (Zheng et al., 2019)            | Online combat-game testing                | State exploration, coverage, and explicit oracles                          | Bug detection in an existing game                                 |
| TITAN (Wang et al., 2025)            | Commercial MMORPG testing                 | State abstraction, action traces, reflection, and LLM oracles              | Diagnostic QA reports                                             |
| Voyager (Wang et al., 2023)          | Minecraft play and skill acquisition      | Environment feedback and executable skills                                 | Open-ended agent learning                                         |
| GCA (ours)                           | End-to-end Phaser game and asset creation | Static, browser, play, layout, visual, and performance evidence            | In-pipeline release gates; separate external evaluation           |

Table 1: Representative game systems and the role of evidence.

**Complete-Game Generation.** GameGPT (Chen et al., 2023) is an early precedent for game-specific role decomposition, organizing development into planning, task classification, code generation, execution, and summarization with reviewer roles. OpenGame (Jiang et al., 2026b) is the closest artifact-level comparison to GCA: it also targets complete Phaser web games from natural-language specifications, separates design from implementation, treats asset synthesis as part of creation, and includes a verification stage. The two differ in where responsibility lives. OpenGame concentrates reusable project and repair knowledge in accumulated skills and a game-specialized code model, whereas GCA partitions files across eight agents and routes failed production checks back to the owning role, keeping its browser, gameplay, layout, visual, and performance evidence inside the

production workflow rather than reporting it as a score. Architectural distinctions are not quality evidence on their own, so Section 5 compares the finished artifacts under one external protocol.

**Game Evaluation and Testing.** Execution-based benchmarks score what an agent builds by running it. GameDevBench (Chi et al., 2026) sets coding agents substantial Godot tasks (Godot Engine Contributors, 2026) and reports lower success as visual complexity increases, and OpenGameBench (Jiang et al., 2026b) executes finished web games in a headless browser, judging build health, visible usability, and prompt alignment apart from one another. GameCraft-Bench (Luo et al., 2026) is the protocol we borrow from most directly: we adopt its four dimensions and its requirement for interaction-conditioned evidence, adapt execution to browser games, and report the dimensions separately. A parallel line tests games rather than building them, from planted-bug exploration (Jiang et al., 2026a) to oracles derived from commercial defects (Zheng et al., 2019) and long-horizon diagnostic agents for commercial titles (Wang et al., 2025), while agents acting through rendered interfaces show how far such exploration can reach (Zhang et al., 2025; He et al., 2024; Zheng et al., 2024; Cheng et al., 2024; Zhou et al., 2024). Voyager explores on a far larger budget, though it acts through a scripting API, accumulating executable skills from textual environment feedback rather than from a rendered screen (Wang et al., 2023). All of them work on a game that already exists, with exploration budgets to match. GCA keeps the same separation between exploration and oracles on a much smaller budget, running bounded checks on a game it has just assembled in order to decide whether the pipeline may advance.

**Content and Asset Generation.** Procedural content generation builds levels and other in-game content from formal or learned models (Shaker et al., 2016; Summerville et al., 2018), and language models extend this to text-conditioned tile-level generation with novelty search for variation (Sudhakaran et al., 2023). Such systems focus on content within a game, whereas complete-game agents must also integrate controls, state transitions, assets, build configuration, and presentation. Latent-diffusion and image-to-video models widen the range of assets a creation pipeline can synthesize (Rombach et al., 2022; Blattmann et al., 2023), but generation alone does not show that an asset is usable in context, and multimodal judges agree only imperfectly with human ratings (Ku et al., 2024). We therefore pair narrow asset and integration rubrics with deterministic size, reference, layout, and runtime checks wherever those exist, and treat visual-model output as one source of evidence rather than as ground truth.

### 3 System Architecture

GCA is an artifact-producing pipeline. Each stage accepts named inputs from its predecessors, writes a class of outputs that it alone owns, and records the evidence its successor requires. A run starts from startup, design, optional authored level design, and scaffold; implementation and asset production then execute concurrently and join before asset integration; and verification alternates with bounded, owner-routed repair until a final full pass clears release. Figure 1 gives this flow, and the rest of the section walks it stage by stage, stating what each stage leaves behind. Sections B and D retain the exact dispatch schemas, gate membership, and thresholds.

One rule governs every transition: a successful agent return does not advance the run (Huang et al., 2024). The orchestrator validates that the promised outputs exist, satisfy their contracts, and belong to the active workspace, pipeline run, and build digest, and only then starts the next stage. It controls dispatch, joins, state transitions, and gates while authoring no game content itself; the substantive work belongs to the eight agents of Table 2, each owning one artifact class. That split

is what makes the artifact graph explicit: the multimodal GDD defines intent, scaffold projects contracts, implementation and asset agents realize disjoint outputs, integration combines them, and verification records evidence about the resulting runtime.

| Agent             | Responsibility                                                                                                                     | Owned artifact class                    |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------|
| design            | Author the multimodal GDD: the design specification plus linked concept mockups per screen and grounded crops of each element      | Design and concept evidence             |
| level-design      | Materialize or revise authored levels; dispatched only when routing calls for them                                                 | Authored level set                      |
| scaffold          | Project the multimodal GDD into a Phaser 4 shell plus the machine-readable contracts and deterministic tests that later gates read | Runtime shell, contracts, and tests     |
| implement         | Implement one scene’s gameplay and its feature tests; dispatched once per scene, serially                                          | Scene implementations and feature tests |
| assets            | Generate and register visual assets for one batch through the control plane, with per-asset metadata sidecars                      | Visual assets and metadata              |
| asset-integration | Wire the finalized manifest into runtime preloads, animations, and display parameters, then seal the integration receipt           | Runtime asset wiring and sealed receipt |
| verify            | Run the verification DAG and fan its producers into one identity-bound evidence report                                             | Verification evidence                   |
| critic            | Judge shippability from evidence alone, having never seen the source code                                                          | Code-blind release verdict              |

Table 2: Specialized agents and their owned artifacts.

**Startup.** The orchestrator establishes the run’s invariants without writing game content. It preserves the request verbatim, selects one game directory, acquires an exclusive workspace lease, derives immutable genre, interaction, quality, and level routing, and persists the run identity. *Produces:* the run identity, the workspace lease, and a routing decision later stages cannot revise. The stage validator therefore rejects files from a previous attempt, and no later image or partial output can change the initial route.

**Design.** The design agent is the sole author of the design specification and its linked visual evidence. Mechanics, state transitions, controls, progression, scene and interface requirements, asset obligations, and testable objectives are paired with screen concepts, normalized layout detections, and grounded element crops. *Produces:* the multimodal GDD, and the ordered scene list the orchestrator extracts from it. The orchestrator validates that the document is complete and that interactive scenes carry concept evidence or an explicit bounded-failure waiver, but it does not repair the design itself. Figure 3 illustrates the relation for the Title, Combat, and Reward scenes of the game GCA produced for the Deckbuilder task of Table 4.

The aligned runtime captures in Figure 3 are downstream outputs with the corresponding initial multimodal GDD. Their visual correspondence traces a scene contract to its implementation, while it says nothing about the game rules like energy, turn order, reward mutation, or deck conservation;

**Exact prompt** “Each turn you draw 5 cards . . . using a limited 3 energy points.”

**[MUST-04]** reward screen: “3 selectable cards and 1 ‘Skip’ button.”

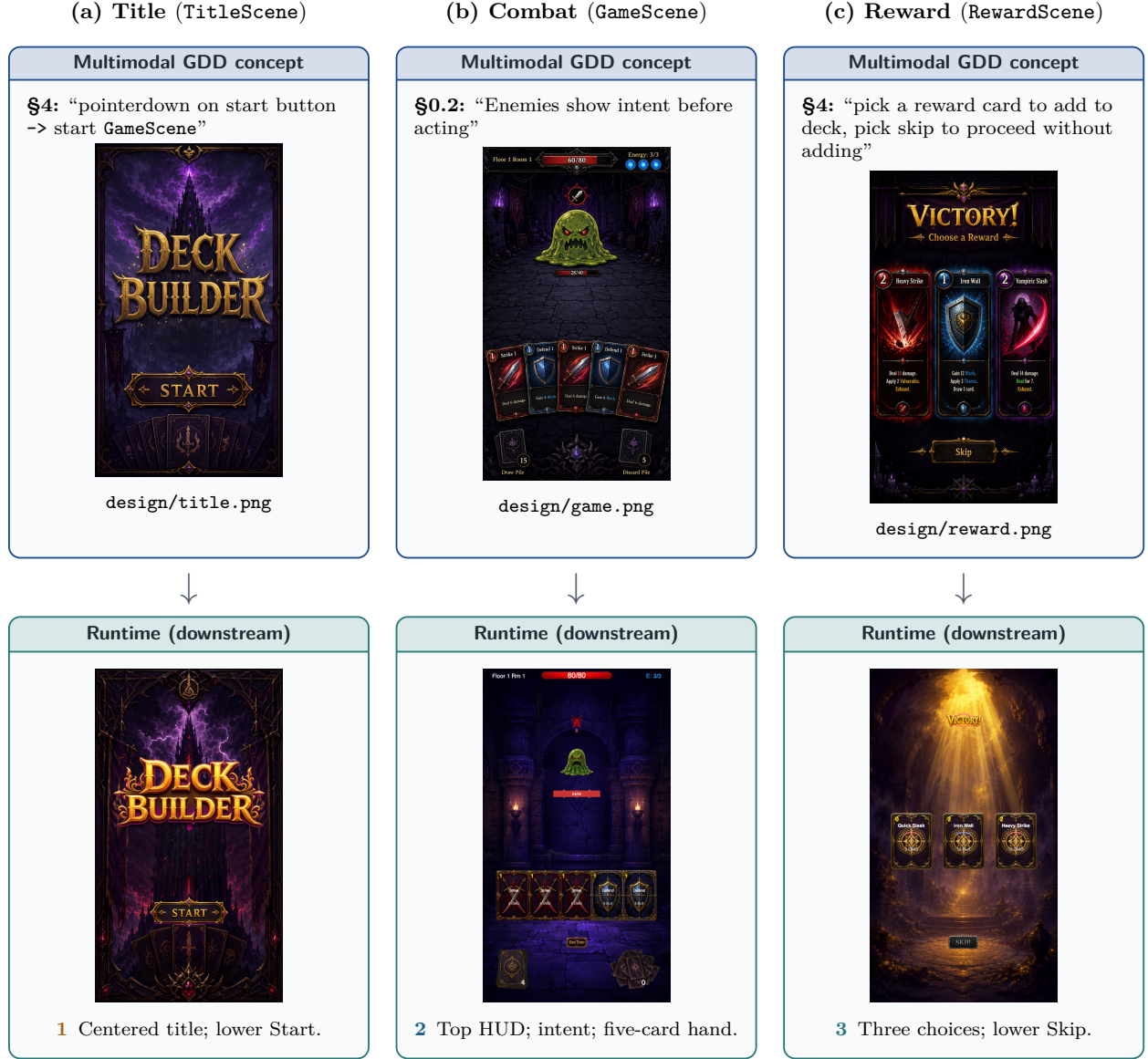


Figure 3: Three Deckbuilder scene contracts link multimodal GDD concepts to matching downstream runtime captures.

those are behavioral claims that only runtime interaction and state checks can settle. Keeping the two apart rejects a plausible visual design with buggy underlying implementation.

**Level Design (Conditional).** If routing selected an authored level model, the level-design agent writes and validates the ordered stage files that progression requires. *Produces:* the level inventory, passed unchanged to scaffold and scene implementation. Procedural, board, and other level-free games skip this stage, and incidental files cannot alter routing.

**Scaffold.** Scaffold projects the accepted multimodal GDD into runnable structure. *Produces:* a Phaser 4 shell, scene stubs, an asset manifest, and machine-readable interaction, objective, layout, rubric, and deterministic test contracts. Because a different role writes them, these contracts are an oracle for code that does not yet exist. A scaffold gate validates the outputs before the server and build branches begin, and if a later design repair changes the authority, scaffold reprojects only the stale paths named in its sealed receipt.

**Scene Implementation.** Implementation runs serially over the declared scene list, because the scenes share a code surface and serialization avoids hidden conflicts between them. *Produces:* scene code and the tests that exercise it.

**Asset Production.** In parallel, independent asset batches are partitioned across asset agents. Workers write owned media and metadata sidecars but cannot touch the shared manifest, and it is that disjoint ownership which makes parallel generation safe. *Produces:* media files, metadata sidecars, and one receipt per batch. Figure 4 shows the bounded mechanism for animated sprites.

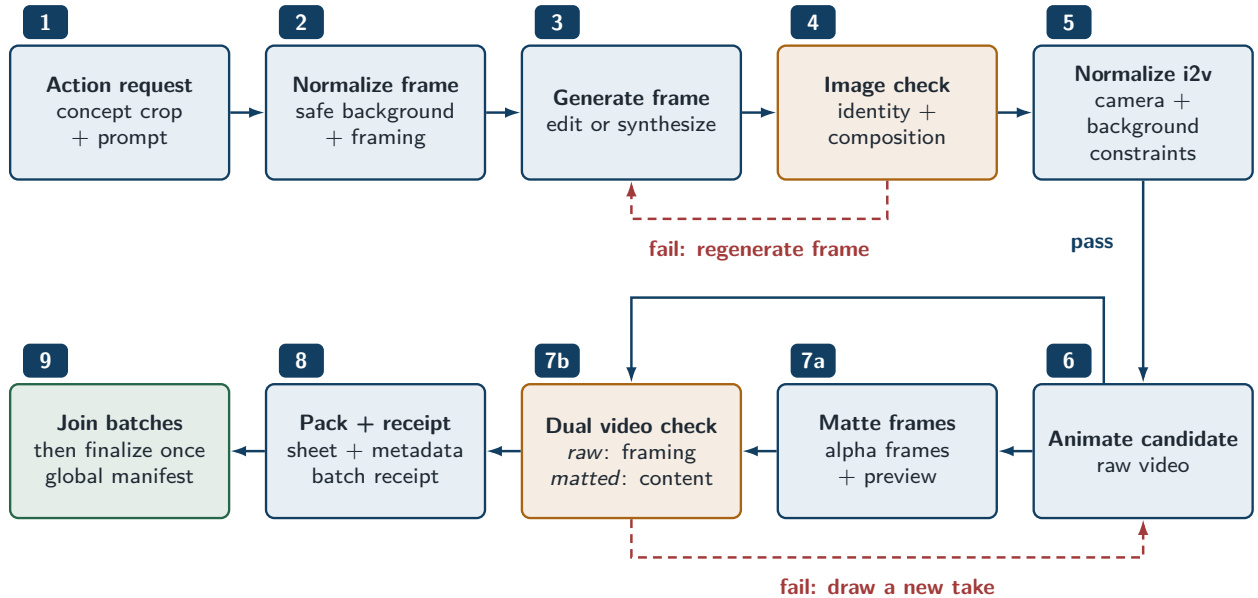


Figure 4: Animated-sprite production alternates bounded generation, checks, retries, and one post-join finalization.

Each entity first receives a shared visual reference. Action-specific generation then works from a grounded concept crop, under fixed framing and background constraints. Raw animation and matted output are checked for framing and shipped-content quality. A failed check may trigger another animation attempt within the configured budget; exhaustion is recorded as best effort rather than relabeled as success. Accepted frames are packed into a Phaser spritesheet with metadata.

**Integration.** The implementation workflow and every declared asset batch join before one reducer merges file-backed sidecars into the manifest. The reducer preserves missing required outputs as failures and records valid no-file sources, such as procedural graphics or live text, explicitly. The asset-integration agent then runs alone, wiring preloads, animations, scaling, display parameters, and interface bindings. *Produces:* the assembled runtime, and a sealed receipt fixing the build digest that

enters verification. The receipt is what keeps multimodal GDD evidence distinct from the runtime assets and code that realize it.

**Verification.** Verification runs on the build the previous stage sealed, and combines deterministic and perceptual evidence without treating the two as interchangeable. Static gates inspect files and contracts; Playwright tests execute interaction rules (Microsoft, 2025); a play agent explores the live game from rendered observations and engine state (Yao et al., 2023; He et al., 2024); layout checks compute bounds and overlap; visual comparisons judge crop-paired appearance (Ku et al., 2024); and a code-blind critic assesses the assembled evidence (Zheng et al., 2023; Gou et al., 2024). *Produces:* a sealed verification report and a gate decision. Figure 5 presents the flow together with one Minesweeper iterate epoch.

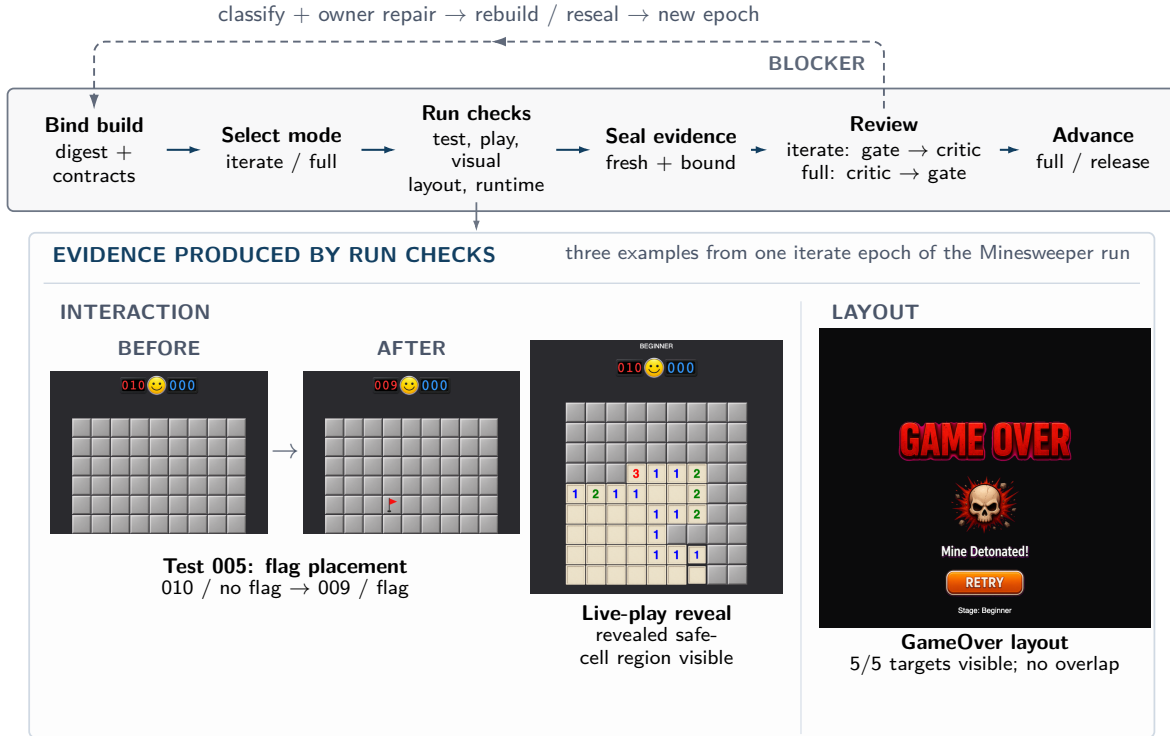


Figure 5: One verification epoch. Checks run against the freshly sealed build and produce the evidence shown here, from a Minesweeper run. The gate and the critic then read the sealed report and either advance the run or return a blocker to its owner for a new epoch.

Every epoch is bound to the current run, build digest, URL, contracts, and verification identity. In **iterate** mode, browser, play, visual, layout, and runtime-asset producers checkpoint their outputs, a finalizer seals the aggregate even if a producer fails, and the iteration gate is evaluated before the mandatory critic. In **full** mode, isolated recordings, video analysis, and performance samples are added, and the final critic precedes the release gate. Both scopes materialize deterministic records before atomic report finalization, and neither accepts evidence from an earlier digest. The Minesweeper epoch of Figure 5 shows this in one run: formal test 005 checks that a real right-click places a flag and decrements the counter, live play checks a safe-cell reveal, and layout capture finds all five GameOverScene targets without overlap or off-screen placement. All three are internal production artifacts, unrelated to the frames and scores of the external experiment.

**Owner-Routed Repair.** When either gate or critic reports a blocker, we group findings by owning agent and route design repairs first (Shinn et al., 2023; Madaan et al., 2023), because they invalidate downstream projections. Code and asset repairs are parallelized only for disjoint write sets; the writers then join, affected shared artifacts are rebuilt, and a new build digest is sealed before another epoch starts. *Produces:* a repaired build under a new digest, or, at budget exhaustion, a preserved record of the unresolved findings.

**Release.** A game ships only once a full verification pass and the final critic clear the release gate under a single build digest. *Produces:* the released game together with the sealed evidence that authorized it. This loop decides whether GCA may advance; it does not determine the external scores in Section 5.

## 4 MCP Control Plane

Every action that makes something or checks something goes through one MCP server (Anthropic, 2024). It is the one place that decides which workspace a call belongs to, whether an artifact has the fields it is supposed to have, and whether the evidence a gate will ask for is already there. No agent has to write those checks itself, and no agent can skip them. Table 3 groups the 55 tools by category; Sections A and B give the full catalog and the exact thresholds.

| Category                          | Tools     | Scope, with a representative tool                                                                                                                                    |
|-----------------------------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Lifecycle                         | 9         | Workspace leasing, preflight readiness, scaffold reproject receipts, and the transactional state journal — <code>write_pipeline_state</code>                         |
| Asset generation and validation   | 20        | Image and video synthesis, matting, spritesheet packing, and the closed-loop producers that retry against VLM rubrics — <code>produce_validated_sprite_action</code> |
| Verification                      | 16        | The checkpointed verification DAG, its individual producers, and the gate runner — <code>run_verify_dag</code>                                                       |
| Concept grounding                 | 3         | Detect element boxes on an approved mockup and crop them into art seeds — <code>detect_layout</code>                                                                 |
| Post-processing and matting       | 3         | CPU compositing plus the two matting backends, remote and deterministic — <code>rmbg2_matting</code>                                                                 |
| Video understanding and recording | 3         | Record a running game with sampled engine state, then analyze only that sealed recording — <code>record_with_state</code>                                            |
| Path and layout                   | 1         | Render a tower-defense path corridor from the level geometry — <code>render_layout_mask</code>                                                                       |
| <b>Total</b>                      | <b>55</b> |                                                                                                                                                                      |

Table 3: MCP control-plane tools by category.

**Typed Tools.** Each tool states what it takes in and what it gives back, right next to the code that runs it and the workspace it needs. Provider calls, browser actions, asset transformations, and gates are all declared the same way. This pays off when several agents run at once: a bad path, a missing identity, or incomplete evidence is refused once, at the call, instead of each agent that runs into it deciding for itself what to make of it.

**Workspace Leases.** Before a tool can touch a workspace, the server pins the directory and takes an exclusive lease on it that other processes have to respect. Two pipelines cannot interleave writes into one game, and nobody can swap the directory out from under a call the server has already accepted. The lease is also where the paper trail starts: everything written after it carries the pipeline run, the build digest, and, where it applies, the verification epoch.

**Stamped Artifacts.** Every result that carries evidence is stamped with the run, build, and epoch it came from. That stamp is how a gate tells a current result from an old file that still looks fine. A report can be complete and internally consistent and still be thrown out, because the run and build it names are not the ones under review.

**Sealed Receipts.** When one agent hands work to the next, the server writes the receipt, not the agent: scaffold reprojection records which paths it invalidated, manifest reduction records which batches it took in, and asset integration records which build it wired. An agent cannot swap one of these receipts for a summary of its own.

**Deterministic Checks and Model Judgments.** Some checks the server can compute; others it can only ask a model to judge, and it keeps the two apart. Bounds, overlap, dimensions, references, counters, and matching identities are computed outright, and asking again gives the same answer. Whether an image looks like its reference, or whether a screen looks finished, is a judgment against a rubric, and pinning the model and its sampling settings makes that judgment repeatable without making it a measurement. The first kind is a derived oracle (Barr et al., 2015); the second is evidence for a person to weigh, not a fact the pipeline can assert.

**Single-Writer Ownership.** The server also gives every artifact exactly one writer, so a quick fix cannot quietly take over someone else’s file. On each write, the boundary hook compares the path against the scope the acting agent declared, across design, levels, scaffolding, scene code, assets, integration, and verification evidence. Figure 6 shows the three outcomes: a write the agent owns goes through, a write into another agent’s file is stopped before the file changes, and the request is handed to the agent that does own it.

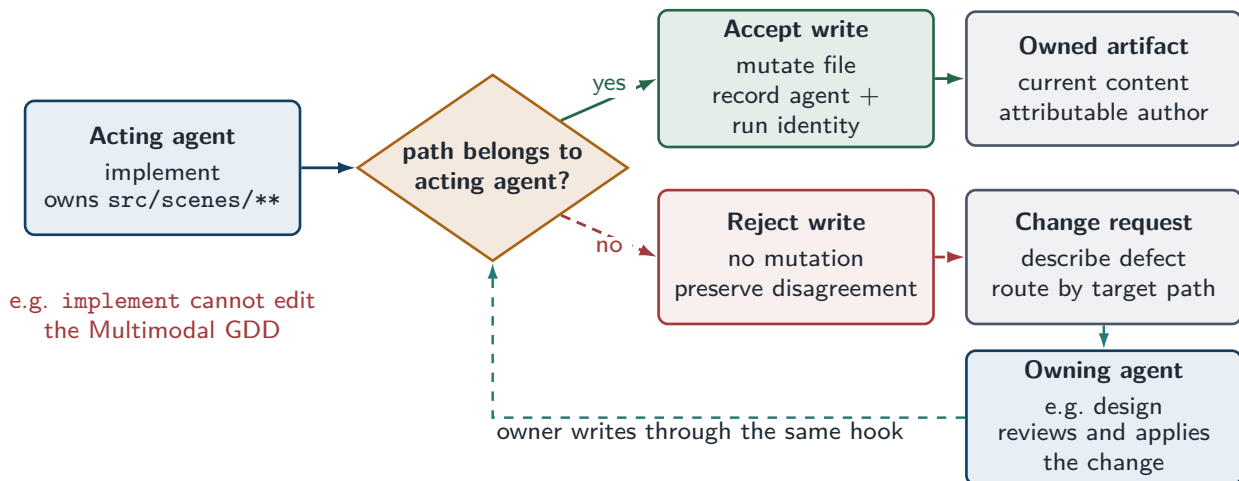


Figure 6: Single-writer mutation control accepts owned writes and routes rejected cross-component changes to the declared owner.

**Recovery.** Restarting after an interruption leans on the same stamps. The orchestrator goes back to the most recent stage boundary whose artifacts, receipts, freshness, and ownership all still hold up, and resumes from there. A leftover marker file does not earn credit for the work it was meant to mark, and an output that is well-formed but stale is refused like any other. Because the server runs these checks on every call, a run that was cut off picks up again without mistaking half-finished writes for finished work.

Typed tools, leases, stamped outputs, receipts, and one writer per artifact are what make a run auditable afterwards. The gates read that evidence to decide what happens next inside the pipeline.

## 5 Experiment

We evaluate finished games independently of GCA’s production controls. Two evaluations run on the same corpus: an automated one that asks how faithfully each system realizes observable prompt requirements, and a human study that rates the same five dimensions and records pairwise preference. The automated evaluator receives only the original prompt and one game, `Evaluator(prompt, game) → scores + evidence`. It receives neither a reference implementation nor GCA’s multimodal GDD, generation trace, gate reports, or critic verdict. The protocol follows the interaction-grounded principle of GameCraft-Bench (Luo et al., 2026) and adapts it to Phaser browser games and the OpenGame baseline (Jiang et al., 2026b).

### 5.1 Setup

**Corpus.** The task pool comprises 32 game briefs carrying 795 numbered MUST requirements. A task enters the experiment when two conditions hold: prompt-matched GCA and OpenGame artifacts exist for it, and a reviewed task-specific browser suite covers its requirements. Twenty tasks satisfy both. Table 4 lists this corpus under report-local sequence numbers 1–20; canonical directory identifiers and source revisions remain in the run manifests. The corpus is fixed before scoring and is not revised afterwards.

| No. | Task           | Family   | No. | Task                 | Family   |
|-----|----------------|----------|-----|----------------------|----------|
| 1   | Minesweeper    | Puzzle   | 11  | Blackjack            | Card     |
| 2   | Snake          | Arcade   | 12  | Endless Runner       | Runner   |
| 3   | Tap Flyer      | Arcade   | 13  | Fruit Slicer         | Action   |
| 4   | Shmup          | Action   | 14  | Solitaire (Klondike) | Card     |
| 5   | Match-3        | Puzzle   | 15  | Link Match           | Puzzle   |
| 6   | Tower Defense  | Strategy | 16  | Gold Miner           | Arcade   |
| 7   | Lemonade Stand | Sim.     | 17  | Basketball Shoot     | Sports   |
| 8   | Platformer     | Platform | 18  | Dawn Dusk Farm       | Sim.     |
| 9   | Deckbuilder    | Card     | 19  | Veinworks            | Idle     |
| 10  | Mole Tap       | Arcade   | 20  | Vibe Fighter         | Fighting |

Table 4: Fixed 20-task paired corpus with game families, numbered in report-local sequence. Tasks 1–10 are listed on the left and 11–20 on the right. Task names describe gameplay genres and mechanics; they do not refer to any commercial game.

**Games and scoring runs.** Each system contributes one game per task, so the experiment compares  $20 \times 2 = 40$  generated artifacts. Every artifact is then scored three times under the same protocol, giving  $40 \times 3 = 120$  selected evaluation runs, 60 per system. The repeats re-score one fixed build rather than regenerate it: deterministic checks return the same verdict on every repeat, and the spread between repeats comes from the model-judged dimensions and from non-determinism in the game itself (Luo et al., 2014). Every selected run completed with full task-specific requirement coverage.

**Controlled factors.** Prompt bytes, suite version, browser configuration, evidence policy, judge, and retry rules are held constant across systems and repeats, so the system under test is the only factor that varies. The fixed settings are Chromium (The Chromium Projects, 2026) at a  $1280 \times 720$  viewport under a 300-second browser ceiling; a full WebM (The WebM Project, 2026) recording, semantic key frames, and 12 uniform video samples as retained evidence; and GPT-5.6 (OpenAI, 2026b) over two passes under a 600-second budget, with at most three attempts per pass for connection failures, rate limits, and retryable server errors. Every score is attributable to the SHA-256 of its prompt, rubric, game tree, and test suite, and credentials are read at runtime and never written into run artifacts.

**Exclusions.** We distinguish a failed run from an invalid measurement. A failed selected run stays in the experiment and is scored as such, whereas an attempt whose measurement is unsound is excluded explicitly rather than recorded as a game failure. Six earlier Solitaire attempts fall in the second category: an evaluator parsing defect invalidated them and valid runs superseded them, so they remain on record as formal exclusions outside the 120 selected evaluation runs.

## 5.2 Automated Evaluation

**Rubric.** Every prompt MUST requirement is preserved as an individual rubric item assigned to Core Mechanics, Content Depth, or Functional Visuals. The harness neither discards nor merges requirements, and it appends three task-independent Art & Presentation items. Missing identifiers, duplicate items, invalid values, or incomplete coverage invalidate the corresponding judge output.

**Evidence.** The evaluator serves each artifact and drives it with Playwright, delivering every action through the rendered interface: clicks, right-clicks, hovers, keyboard input, and drag gestures. Source, Document Object Model, layout, and public runtime state may be read to locate rendered targets and assert outcomes, but the suite may not invoke an internal action or mutate state in place of user input. Tests cover controls, rules, progression, terminal states, restart behavior, and task-specific requirements while retaining console, page, and request errors. Each run stores scenario screenshots and a complete gameplay recording; semantic frames are associated with named scenarios and are not assumed to form a single chronology when scenarios initialize independently. GPT-5.6 receives the prompt, rubric, deterministic assertion record, key screenshots, and time-ordered frames sampled from the recording, and an initial judgment is followed by an audit pass that rechecks every requirement, failed assertion, terminal condition, and cited visual observation.

**Scoring.** Table 5 defines the five reported quantities. All item scores lie on a continuous  $[0, 1]$  scale, and missing values remain unmeasured rather than becoming zero. Core Mechanics and Content Depth are weighted fractions of deterministic real-UI assertions that pass. Functional Visuals instead asks whether required state, interface values, action feedback, and outcomes are visible in rendered evidence: under the grounded-visual scoring policy, GPT-5.6 assigns each prompt-derived item

directly from screenshots and video frames while receiving deterministic assertions as factual context. A failed assertion must be acknowledged in the rationale and contradiction provenance, but the assertion score is not a second numeric operand, which preserves the distinction between an exact behavioral check and an evidence-grounded visual judgment. GPT-5.6 scores Art & Presentation from the same visual evidence using three shared items: visual coherence, composition and readability, and authored finish. Requirement Satisfaction is the mean of all prompt-derived Core Mechanics, Content Depth, and Functional Visuals items; Art & Presentation is excluded because it measures general presentation rather than a prompt MUST requirement. We compute no aggregate overall quality score.

**Results.** We report a scoring-only derivation from immutable earlier run manifests, browser evidence, deterministic records, and stored GPT-5.6 item judgments. It made no new model request, and versioned records hash-link their source artifacts. Table 6 gives the aggregate figures and the

| Metric                   | Output | Rubric question                                                             | Scoring                                                                                 |
|--------------------------|--------|-----------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| Requirement Satisfaction | [0, 1] | What fraction of the observable prompt MUST requirements is satisfied?      | Mean of all prompt-derived Core Mechanics, Content Depth, and Functional Visuals items. |
| Core Mechanics           | [0, 1] | Do controls, rules, state transitions, outcomes, and restart behavior work? | Weighted fraction of passed deterministic MUST-item checks.                             |
| Content Depth            | [0, 1] | Are requested modes, levels, progression, and terminal content present?     | Weighted fraction of passed deterministic MUST-item checks.                             |
| Functional Visuals       | [0, 1] | Are required game state, HUD, action feedback, and outcomes visible?        | Mean of direct evidence-grounded GPT-5.6 MUST-item scores.                              |
| Art & Presentation       | [0, 1] | Is the presentation coherent, readable, and visibly finished?               | Mean of three shared GPT-5.6 visual-quality items.                                      |

Table 5: Test metrics and scoring rubric.

| Metric                   | Mean $\pm$ standard deviation       |                                     |
|--------------------------|-------------------------------------|-------------------------------------|
|                          | GCA                                 | OpenGame                            |
| Requirement Satisfaction | <b>0.786 <math>\pm</math> 0.151</b> | 0.656 $\pm$ 0.247                   |
| Core Mechanics           | <b>0.852 <math>\pm</math> 0.144</b> | 0.655 $\pm$ 0.267                   |
| Content Depth            | <b>0.712 <math>\pm</math> 0.286</b> | 0.651 $\pm$ 0.267                   |
| Functional Visuals       | <b>0.711 <math>\pm</math> 0.192</b> | 0.667 $\pm$ 0.248                   |
| Art & Presentation       | 0.797 $\pm$ 0.099                   | <b>0.827 <math>\pm</math> 0.068</b> |

Table 6: Aggregate prompt-grounded results. Each system contributes one game per task and every game is scored three times, so a cell averages 60 evaluation runs over 20 games and its deviation spans tasks as well as scoring repeats. Bold marks the higher mean in each row. Core Mechanics and Content Depth derive from deterministic checks, Functional Visuals and Art & Presentation from grounded GPT-5.6 judgments, and Requirement Satisfaction averages the prompt-derived items across the first three dimensions. Null values are excluded, not converted to zero.

rules used to form them. GCA has higher mean scores on Requirement Satisfaction (0.786 versus 0.656), Core Mechanics (0.852 versus 0.655), Content Depth (0.712 versus 0.651), and Functional Visuals (0.711 versus 0.667), while OpenGame scores higher on Art & Presentation (0.827 versus 0.797). Each value is computed over the 60 evaluation runs of a system’s 20 games under a common protocol, so its spread reflects differences between tasks as well as between scoring repeats of one game. Task-level directions vary, particularly for Content Depth and Functional Visuals.

**Audit and caveats.** Relative to the earlier merged scoring, the derivation changed 78 Functional Visuals item scores across 41 runs and audited the 254 items tied to at least one failed assertion. Its source judgments predate a stricter frame-citation contract: 569 of 600 visual items cite an exact frame label or a child check of a frame-labeled capture, while 31 cite scenario or assertion records even though they received the complete frame bundle. Current judgments require an exact rendered-frame reference for every Functional Visuals and Art & Presentation item. The caveat therefore concerns how those 31 items cite their evidence; corpus coverage is unaffected.

### 5.3 Human Study

**Protocol.** Ten participants each evaluated all 20 prompt-matched pairs, yielding 200 participant–game ratings per system and 200 pairwise preference choices. System identity was hidden behind randomized Game A/Game B labels, and presentation order was counterbalanced. For each pair, participants rated both games on anchored five-level versions of Requirement Satisfaction, Core Mechanics, Content Depth, Functional Visuals, and Art & Presentation, then selected Game A, Game B, or no preference. Preference remained categorical and was not converted to the rating scale. Section E provides the one-page form completed once for each of a participant’s 20 pairs.

**Results.** Table 7 gives the rubric means with their standard deviations and the preference shares. GCA received higher mean ratings on all five dimensions, with the largest gap in Art & Presentation (3.57 versus 3.03). Participants preferred GCA in 86 of 200 pair choices, OpenGame in 55, and

| Reported quantity                                                | GCA                | OpenGame    |
|------------------------------------------------------------------|--------------------|-------------|
| <i>Rubric ratings on a 1–5 scale (mean ± standard deviation)</i> |                    |             |
| Requirement Satisfaction                                         | <b>3.55 ± 1.36</b> | 3.28 ± 1.47 |
| Core Mechanics                                                   | <b>3.40 ± 1.31</b> | 3.16 ± 1.36 |
| Content Depth                                                    | <b>3.43 ± 1.36</b> | 3.28 ± 1.40 |
| Functional Visuals                                               | <b>3.33 ± 1.35</b> | 3.29 ± 1.33 |
| Art & Presentation                                               | <b>3.57 ± 1.13</b> | 3.03 ± 1.19 |
| Overall                                                          | <b>3.45 ± 1.13</b> | 3.21 ± 1.18 |
| <i>Pairwise preference, share of choices (%)</i>                 |                    |             |
| All 200 choices                                                  | <b>43.0</b>        | 27.5        |
| Excluding ties                                                   | <b>61.0</b>        | 39.0        |
| No preference                                                    |                    | 29.5        |

Table 7: Human study on 20 prompt pairs. Ten participants each rated all 20 pairs, giving 200 participant–game ratings per system. Bold marks the higher value in each row, and *Overall* is the mean of the five dimensions. Of the 200 preference choices 59 were “no preference”; excluding ties, GCA is preferred in 86 of 141 decisive choices (two-sided sign test,  $p = 0.011$ ).

expressed no preference in 59. Among the 141 decisive choices, the preference split was 61.0% versus 39.0% in favor of GCA (two-sided sign test,  $p = 0.011$ ). The ratings complement the automated results without standing in for their prompt-derived, interaction-grounded evidence.

#### 5.4 Case Study: Gold Miner

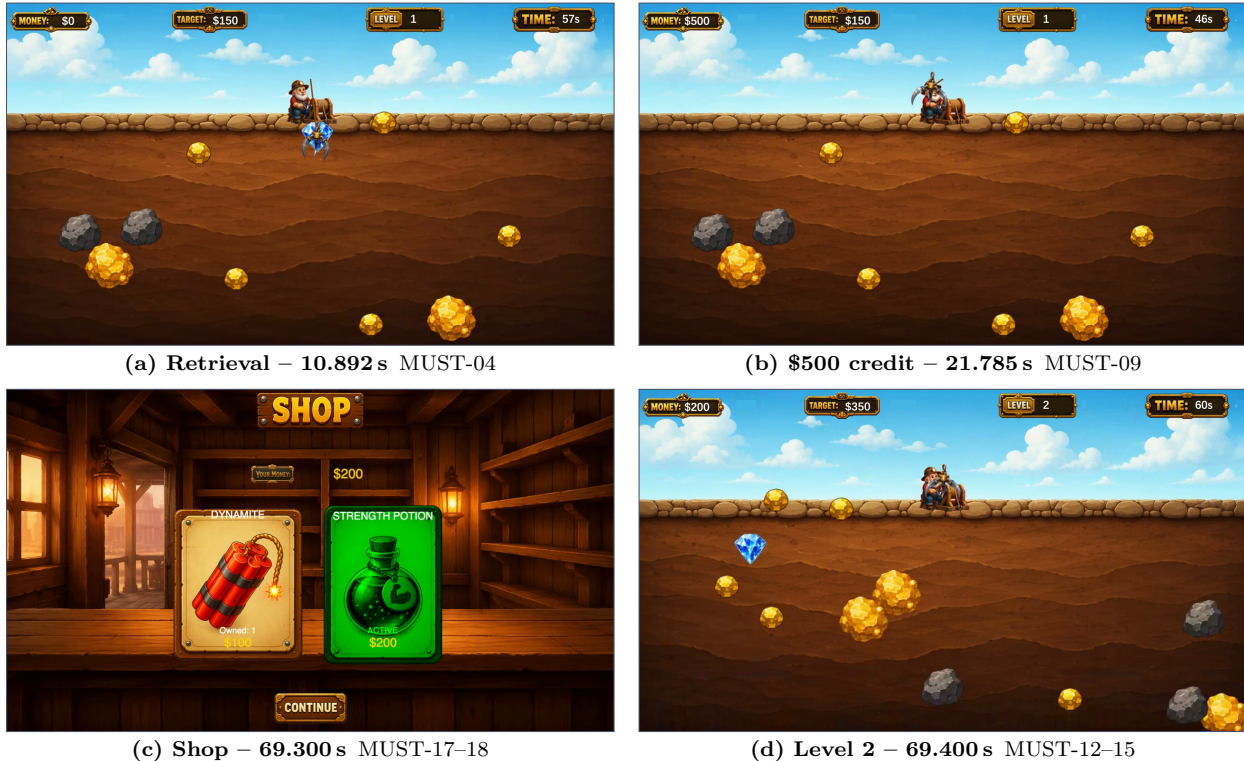


Figure 7: Gold Miner formal-run evidence: retrieval, exact \$500 credit, shop purchase, and level-2 transition in one continuous recording.

We use Gold Miner to connect the architecture to a formal external run while preserving the evidence boundary. Its prompt asks for claw retrieval, mineral values, a countdown timer, cumulative money, per-level targets, shop items, runtime states, and interface readouts. The multimodal GDD maps those requirements to controls, transitions, scenes, and screen concepts, and scaffold projects the scene, input, objective, layout, and observability contracts.

The internal and external records disagree about this run, and that is the point of reporting it. The pipeline ended blocked rather than released: a timing-sensitive launch guard remained, visual-comparison provider calls failed, and metadata checks were unresolved. The external suite, driving the rendered interface with runtime state read only for assertions, observed a coherent prefix of the requested game. Figure 7 shows four frames from one continuous recording, covering retrieval, the exact credit after collection and removal, a shop purchase, and the transition into the next level with the balance carried over. Support is partial for claw-tip collision, final victory, and fresh restart, and collection is inconsistent across the other scoring repeats.

Neither record subsumes the other. The gates withheld release on evidence the external suite never sees, while the external suite confirmed prompt fulfillment that a blocked internal status would otherwise hide. Reading the two separately is what keeps each of them meaningful.

## 6 Conclusion

We presented GCA, a pipeline that turns one natural-language request into a complete, playable Phaser browser game: a multimodal GDD grounds design intent in scene evidence, scaffold projects that document into a runtime shell with machine-readable contracts and deterministic tests, eight specialized agents implement scenes serially while generating assets in parallel, and a typed MCP control plane binds every artifact, receipt, and item of evidence to one run and build digest. To decide whether a game is done, GCA tests the build it just assembled instead of trusting what an agent says it did. That is why it can run with nobody watching and still refuse to ship a game that only looks finished. Judged with rendered interfaces, the games it produces satisfy prompt requirements and implement working mechanics at a level comparable open pipelines do not reach.

## Contributors

**Core contributors.** Xiangxin Zhou, Jiaqi Tang, Ke Ma, Wenhao Yang, Jiarui Yao, Boye Niu, Haiyang Guo, Yifei Zhao, Huanjin Yao, Tianyu Pang.

**Other contributors.** Pu Jian, Ruihuang Li, Haitao Wu, Yushi Huang, Xiangyan Liu, Yan Sun, Kaixin Li.

## References

- Anthropic. Model context protocol. <https://modelcontextprotocol.io>, 2024.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.
- Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, 2015.
- Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, et al. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv preprint arXiv:2311.15127*, 2023.
- BRIA AI. RMBG-2.0: Background removal model. <https://huggingface.co/briaai/RMBG-2.0>, 2024.
- ByteDance Seed Vision Team. Seedance 1.0: Exploring the boundaries of video generation models. *arXiv preprint arXiv:2506.09113*, 2025.
- Dake Chen, Hanbin Wang, Yunhao Huo, Yuzhao Li, and Haoyang Zhang. GameGPT: Multi-agent collaborative framework for game development. *arXiv preprint arXiv:2310.08067*, 2023.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.
- Wayne Chi, Yixiong Fang, Arnav Yayavaram, Siddharth Yayavaram, Seth Karten, Qihong Anna Wei, Runkun Chen, Alexander Wang, Valerie Chen, Ameet Talwalkar, and Chris Donahue. GameDevBench: Evaluating agentic capabilities through game development. In *International Conference on Machine Learning (ICML)*, 2026.
- Godot Engine Contributors. Godot engine: A free and open source 2D and 3D game engine. <https://godotengine.org/>, 2026.
- Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations (ICLR)*, 2024.
- Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024.

- Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations (ICLR)*, 2024.
- Shufan Jiang, Chios Chen, and Zhiyang Chen. GBQA: A game benchmark for evaluating LLMs as quality assurance engineers. *arXiv preprint arXiv:2604.02648*, 2026a.
- Yilei Jiang, Jinyuan Hu, Qianyin Xiao, Yaozhi Zheng, Ruize Ma, Kaituo Feng, Jiaming Han, Tianshuo Peng, Kaixuan Fan, Manyuan Zhang, and Xiangyu Yue. OpenGame: Open agentic coding for games. *arXiv preprint arXiv:2604.18394*, 2026b.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations (ICLR)*, 2024.
- Max Ku, Dongfu Jiang, Cong Wei, Xiang Yue, and Wenhua Chen. VIEScore: Towards explainable metrics for conditional image synthesis evaluation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.
- Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. An empirical analysis of flaky tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 643–653, 2014.
- Tongxu Luo, Rongsheng Wang, Jiayi Bi, Chenming Xu, Zhengyang Tang, et al. GameCraft-Bench: Can agents build playable games end-to-end in a real game engine? *arXiv preprint arXiv:2606.17861*, 2026.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- Microsoft. Playwright: Fast and reliable end-to-end testing for modern web apps. <https://playwright.dev/>, 2025.
- OpenAI. GPT-Image-2. <https://developers.openai.com/api/docs/models/gpt-image-2>, 2026a.
- OpenAI. GPT-5.6. <https://developers.openai.com/api/docs/models/gpt-5.6>, 2026b.
- Phaser Studio. Phaser: A fast, free and fun open source framework for canvas and WebGL powered browser games. <https://phaser.io/>, 2025.
- Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.
- Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

- Noor Shaker, Julian Togelius, and Mark J Nelson. *Procedural Content Generation in Games*. Springer, 2016.
- Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- Shyam Sudhakaran, Miguel González-Duque, Matthias Freiberger, Claire Glanois, Elias Najarro, and Sebastian Risi. MarioGPT: Open-ended text2level generation through large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games*, 10(3):257–270, 2018.
- The Chromium Projects. Chromium: An open-source browser project. <https://www.chromium.org/Home/>, 2026.
- The WebM Project. WebM: An open web media project. <https://www.webmproject.org/>, 2026.
- Chengjia Wang, Lanling Tang, Ming Yuan, Jiongchi Yu, Xiaofei Xie, and Jiajun Bu. Leveraging LLM agents for automated video game testing. *arXiv preprint arXiv:2509.22170*, 2025.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *Conference on Language Modeling (COLM)*, 2024.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- Alex L Zhang, Thomas L Griffiths, Karthik R Narasimhan, and Ofir Press. VideoGameBench: Can vision-language models complete popular video games? *arXiv preprint arXiv:2505.18134*, 2025.
- Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4V(ision) is a generalist web agent, if grounded. In *International Conference on Machine Learning (ICML)*, 2024.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2023.
- Yan Zheng, Xiaofei Xie, Ting Su, Lei Ma, Jianye Hao, Zhaopeng Meng, Yang Liu, Ruimin Shen, Yinfeng Chen, and Changjie Fan. Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2024.

The appendix is the system reference behind the main text. Appendix A catalogs every tool on the MCP control plane. Appendix B lists every gate, the membership of each composite gate, the gate ladder, and the complete threshold reference. Appendix C gives the VLM rubrics used for asset acceptance and release review. Appendix D specifies the agent dispatch interfaces. Appendix E provides the one-page questionnaire repeated for each of a participant’s 20 assigned pairs.

## Appendix A. MCP Tool Reference

This is the complete control plane, enumerated from the live tool registry instead of from documentation. All tools are exposed over the Model Context Protocol (Anthropic, 2024). The **W** column marks tools that declare `requires_workspace` and therefore refuse to run outside a leased game workspace; the dispatcher enforces this centrally, so no handler re-checks it. External back-ends appear with the tools that wrap them: GPT-Image-2 (OpenAI, 2026a) for image synthesis, Seedance (ByteDance Seed Vision Team, 2025) for animation, RMBG-2.0 (BRIA AI, 2024) for matting, and Playwright (Microsoft, 2025) for browser work. The generated game runs on Phaser (Phaser Studio, 2025).

### A.1 Lifecycle (9)

Table 8: Lifecycle tools.

| Tool                        | W | Purpose                                                                                                                                                |
|-----------------------------|---|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| ping                        |   | Health check: workspace and credential status.                                                                                                         |
| init_workspace              |   | Open one game directory under explicit <code>create</code> or <code>resume</code> intent; pins the inode and takes an exclusive repository-wide lease. |
| close_workspace             |   | Detach the workspace and release its lease; gated tools then fail.                                                                                     |
| list_tools_summary          |   | One-line summary of every registered tool.                                                                                                             |
| preflight_check             |   | Startup readiness: credentials, R2 upload config, filesystem, Playwright, ffmpeg/ffprobe, matting endpoint.                                            |
| asset_pipeline_smoke_test   | ✓ | End-to-end probe before asset work: tiny image upload, matting, video credentials.                                                                     |
| seal_scaffold_reproject     | ✓ | Seal the scaffold-declared stale path list into a receipt; callers cannot supply or broaden the paths.                                                 |
| delete_scaffold_owned_paths | ✓ | Consume exactly one sealed receipt and delete its obsolete scene, helper, and test files.                                                              |
| write_pipeline_state        | ✓ | Transactional journal and write of <code>state.json</code> and <code>resume.json</code> ; preserves run identity, enforces monotonic counters.         |

### A.2 Asset Generation and Validation (20)

Table 9: Asset-generation and validation tools.

| Tool                                         | W | Purpose                                                                                                                  |
|----------------------------------------------|---|--------------------------------------------------------------------------------------------------------------------------|
| <code>generate_image</code>                  | ✓ | Text-to-image; wraps the prompt in a chroma-key background constraint for mappable asset types.                          |
| <code>edit_image</code>                      | ✓ | Reference-conditioned edit; derives a game-ready sprite from an approved concept crop.                                   |
| <code>generate_seedance_video</code>         | ✓ | Image-to-video or text-to-video on the active provider channel.                                                          |
| <code>extend_background</code>               | ✓ | Seamless rightward outpainting for static backgrounds.                                                                   |
| <code>build_i2v_prompt</code>                |   | Wrap an animation prompt in the validated constraint block: background lock, hard-edged effects, no glow or fade.        |
| <code>rmbg2_video_matting</code>             | ✓ | Per-frame matting to an RGBA sequence plus encoded transparent outputs.                                                  |
| <code>validate_asset_video</code>            | ✓ | Rubric check on an animation, run against the raw or matted video depending on rubric group.                             |
| <code>produce_validated_video</code>         | ✓ | Generate, matte, and two-pass validate an animation, retrying within configured bounds.                                  |
| <code>validate_image</code>                  | ✓ | Rubric check on a sprite first frame; the pre-animation gate.                                                            |
| <code>validate_asset_slot</code>             | ✓ | Slot-fit check of a finished runtime artifact against its design slot; batchable across the manifest.                    |
| <code>produce_validated_first_frame</code>   | ✓ | Closed generation-validation loop, retrying on failure with validator feedback.                                          |
| <code>produce_validated_sprite_action</code> | ✓ | Full first-frame, animation, and matte chain in one call, with per-stage validation strategies.                          |
| <code>analyze_sprite_metadata</code>         | ✓ | Content bounding box from the alpha channel plus inferred facing.                                                        |
| <code>pack_spritesheet</code>                | ✓ | Pack matted frames into a Phaser sheet and JSON sidecar under the texture-size limit.                                    |
| <code>register_existing_static_asset</code>  | ✓ | Seal provenance for a supplied or licensed PNG without regenerating it.                                                  |
| <code>finalize_asset_manifest</code>         | ✓ | Single-writer post-join reducer; re-opens every file and validates measured dimensions against the manifest.             |
| <code>content_bbox</code>                    | ✓ | Opaque-content bounds of a static asset for collision and display metadata.                                              |
| <code>trim_transparent</code>                | ✓ | Crop a static asset to its alpha content box.                                                                            |
| <code>scale_uniform</code>                   | ✓ | Aspect-preserving resize to the authored in-game display size.                                                           |
| <code>produce_static_asset</code>            | ✓ | Recommended default for static props and UI: generation, matting, trimming, scaling, and sealed publication in one call. |

### A.3 Verification (16)

Table 10: Verification tools.

| Tool                                        | W | Purpose                                                                                                                                |
|---------------------------------------------|---|----------------------------------------------------------------------------------------------------------------------------------------|
| <code>run_verify_dag</code>                 | ✓ | Canonical pass: opens or resumes an epoch, runs producers concurrently, fans in, finalizes in a <code>finally</code> block, and gates. |
| <code>begin_verify_run</code>               | ✓ | Low-level epoch open under the workspace lock; diagnostics and recovery only.                                                          |
| <code>run_tests</code>                      | ✓ | Playwright suite against the epoch URL; a runtime token binds the browser to this workspace.                                           |
| <code>play_game</code>                      | ✓ | Closed-loop play agent scoring itself against the executable objectives specification.                                                 |
| <code>visual_diff</code>                    | ✓ | Whole-screen and per-element perceptual comparison against concept mockups and crops.                                                  |
| <code>capture_layout</code>                 | ✓ | Deterministic capture of rendered object bounds for the layout axis.                                                                   |
| <code>capture_runtime_assets</code>         | ✓ | Capture loaded texture keys, animations, and display-list identity from the running game.                                              |
| <code>materialize_gameplay_alignment</code> | ✓ | Project the play report onto the gameplay axis.                                                                                        |
| <code>materialize_physics_alignment</code>  | ✓ | Project body, art, and world checks onto the physics axis, or mark it not applicable.                                                  |
| <code>materialize_layout_alignment</code>   | ✓ | Non-blocking evaluation of the layout axis from captured bounds.                                                                       |
| <code>write_asset_integration_report</code> | ✓ | Seal the manifest, file, and runtime-wiring evidence produced by integration.                                                          |
| <code>write_verify_report</code>            | ✓ | Fan producer receipts into the epoch’s aggregate report.                                                                               |
| <code>gate_check</code>                     | ✓ | Run a named gate; returns <code>blocked</code> plus routed <code>failures[]</code> .                                                   |
| <code>check_ui_contract</code>              | ✓ | Advisory standalone UI contract diagnostic; not a release authority.                                                                   |
| <code>check_td_contract</code>              | ✓ | Advisory tower-defense structural diagnostic.                                                                                          |
| <code>check_interaction_contract</code>     | ✓ | Advisory collision and interaction diagnostic.                                                                                         |

#### A.4 Concept Grounding, Post-Processing, Video, and Layout (10)

Table 11: Concept-grounding, post-processing, video, and layout tools.

| Tool                         | W | Purpose                                                                                            |
|------------------------------|---|----------------------------------------------------------------------------------------------------|
| <i>Concept grounding (3)</i> |   |                                                                                                    |
| <code>detect_layout</code>   |   | Precision vision pass: tight canvas-relative boxes with roles and anchors for a mockup’s elements. |
| <code>ground_crop</code>     |   | Cut detected boxes into per-element reference images that seed asset generation.                   |

| Tool                                         | W | Purpose                                                                                                               |
|----------------------------------------------|---|-----------------------------------------------------------------------------------------------------------------------|
| <code>fit_image_to_resolution</code>         |   | Fit a concept mockup to the exact game resolution; design files only, never runtime assets.                           |
| <i>Post-processing and matting (3)</i>       |   |                                                                                                                       |
| <code>rmbg2_matting</code>                   | ✓ | Remote soft-alpha background removal; the preferred backend.                                                          |
| <code>color_key_matting</code>               | ✓ | Deterministic CPU chroma-key removal; the outage fallback.                                                            |
| <code>postprocess</code>                     | ✓ | CPU utilities: layered composite previews and seamless tiling.                                                        |
| <i>Video understanding and recording (3)</i> |   |                                                                                                                       |
| <code>record_with_state</code>               | ✓ | Record the running game while sampling engine state; emits video, state timeline, performance summary, and a receipt. |
| <code>analyze_video</code>                   | ✓ | Analyze only a recording sealed by the current epoch's receipt; rejects foreign or hash-mismatched video.             |
| <code>get_state_extractor</code>             |   | Return the bundled state-extractor module that scaffolded tests import.                                               |
| <i>Path and layout (1)</i>                   |   |                                                                                                                       |
| <code>render_layout_mask</code>              | ✓ | Render a tower-defense path-corridor mask from the level geometry source of truth.                                    |

## Appendix B. Gate and Threshold Reference

### B.1 Composite Gate Membership

Besides the four composites below and the `step_done` phase diagnostic, every registry name is an individual check, callable standalone for diagnosis; `levels` fans out to its own per-level checks, and three emit JSON rather than a verdict (`build_digest`, `gate_contract`, `materialize_layout_alignment`).

`scaffold_done` runs 7 checks: `basic_structure`, `pipeline_state`, `levels`, `scaffold_contracts`, `test_contract`, `rubric_contract`, and `asset_manifest`. It is not a subset of the release chain: `pipeline_state` and `scaffold_contracts` appear in no other composite, because the projected contracts they guard are superseded once implementation begins.

`iteration_ready` runs the 32 checks in Table 12 with the verification report restricted to `iterate` mode; genre-specific checks return not applicable outside their genre. `first_pass_complete` runs the same 32 under `full` or `full-novideo` and adds `release_waivers`, `playthrough_recorded`, and `performance`, with 35 checks in total. `ready_to_ship` adds `no_open_issues`, `critic`, `polish_rounds`, and `play_quality`, for 39 checks; `gate_contract` machine-checks the superset relation.

| Group                   | Checks                                                                                                                                                                                                                     |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure and contracts | <code>basic_structure</code> , <code>levels</code> , <code>level_runtime</code> , <code>test_contract</code> , <code>rubric_contract</code> , <code>asset_manifest</code> , <code>asset_runtime_references</code>          |
| Build completion        | <code>scenes_implemented</code> , <code>assets_generated</code> , <code>asset_integration</code>                                                                                                                           |
| Execution evidence      | <code>tests_passing</code> , <code>test_coverage</code> , <code>verify_report</code> , <code>visual_verified</code>                                                                                                        |
| Alignment axes          | <code>visual_alignment</code> , <code>gameplay_alignment</code> , <code>physics_alignment</code> , <code>element_alignment</code> , <code>layout_alignment</code> , <code>visual_diff</code> , <code>scene_coverage</code> |
| Asset integrity         | <code>bbox_coverage</code> , <code>crop_coverage</code> , <code>aspect_drift</code> , <code>matting_complete</code> , <code>no_fx_spritesheets</code> , <code>subject_margin</code> , <code>slot_validation</code>         |
| Genre and interaction   | <code>interaction_contract</code> , <code>ui_contract</code> , <code>ui_layout</code> , <code>tower_defence_contract</code>                                                                                                |

Table 12: Checks in `iteration_ready`, grouped by function.

`step_done` is cumulative over steps 2 through 7. It always runs `basic_structure` and `pipeline_state`; step 2 adds the five scaffold-boundary checks, step 3 `level_runtime`, `scenes_implemented`, and `assets_generated`, step 4 `asset_integration`, `entity_scale`, `ui_proportion`, and `slot_validation`, step 5 nothing (the verification entry boundary), step 6 `visual_verified`, and step 7 `release_waivers`, `playthrough_recorded`, `verify_report` in a full mode, and `performance`, for 19.

### B.2 The Gate Ladder

Figure 8 draws the four composites to scale. They sit at distinct cost boundaries: `scaffold_done` after scaffolding, `iteration_ready` on every fix round, `first_pass_complete` at the first full-mode pass, and `ready_to_ship` only on release. Each is all-or-nothing — no partial credit, no aggregate score to optimize — and when a repair or visual budget is exhausted the pipeline stops with the unresolved list rather than continuing unbounded.

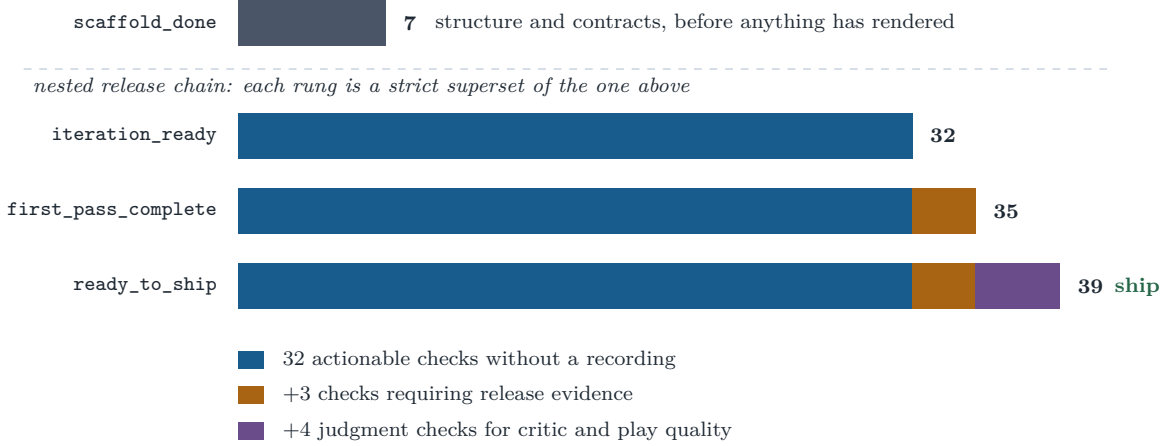


Figure 8: Composite gate sizes, from the seven-check scaffold boundary to the 39-check release gate.

### B.3 Threshold Reference

All values below are the defaults in `gca.config.yaml`, the sole authority: no agent specification, gate implementation, or reference document restates them.

Table 13: Default pipeline, gate, provider, and verification thresholds.

| Key                                                          | Value     | Effect                                    |
|--------------------------------------------------------------|-----------|-------------------------------------------|
| <i>Iteration budget and evidence freshness (pipeline)</i>    |           |                                           |
| <code>min_iterations / max_iterations</code>                 | 1 / 2     | Required fix rounds, and the hard cap     |
| <code>max_consecutive_blocks</code>                          | 5         | Safety valve ahead of the budget          |
| <code>min_screenshots / min_unique_screenshots</code>        | 3 / 3     | Distinct screenshots required as evidence |
| <code>staleness_tolerance_seconds</code>                     | 60        | Clock-skew slack on freshness             |
| <code>max_analysis_age_minutes</code>                        | 30        | Analyses expire after this                |
| <code>min_iteration_gap_seconds</code>                       | 120       | Minimum spacing between iterations        |
| <code>max_counter_increment</code>                           | 1         | Guarded counters advance by at most one   |
| <code>audit_tolerance</code>                                 | 1         | Slack between audit events and counters   |
| <i>Play, polish, and rubric floors (gates, gates.rubric)</i> |           |                                           |
| <code>play_quality_min_accuracy</code>                       | 0.8       | Play-agent weighted accuracy floor        |
| <code>polish.visual_rounds_min</code>                        | 1         | Visual evidence rounds at release         |
| <code>polish.gameplay_rounds_min</code>                      | 1         | Gameplay evidence rounds at release       |
| <code>polish.require_stress_tested</code>                    | false     | Optional stress evidence                  |
| <code>intention_min</code>                                   | 0.9       | Does the video show the intended game     |
| <code>visual_min / interaction_min</code>                    | 0.7 / 0.8 | Visual quality and interaction legibility |
| <i>Performance (gates.performance)</i>                       |           |                                           |
| <code>min_fps_p50 / min_fps_min</code>                       | 24 / 10   | Frame-rate floors                         |

| Key                                                                                                                                            | Value                     | Effect                                                                          |
|------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|---------------------------------------------------------------------------------|
| <code>max_boot_seconds</code>                                                                                                                  | 10                        | Boot budget                                                                     |
| <code>max_long_task_count</code>                                                                                                               | 20                        | Long-task occurrences                                                           |
| <code>max_long_task_total_ms</code>                                                                                                            | 2000                      | Long-task total                                                                 |
| <code>max_memory_growth_mb</code>                                                                                                              | 50                        | Memory-leak detector                                                            |
| <code>max_object_growth_count</code>                                                                                                           | 250                       | Entity-leak detector                                                            |
| <code>max_object_slope_count_per_second</code>                                                                                                 | 10                        | Leak growth rate                                                                |
| <i>Visual loop</i> ( <code>gates.visual</code> )                                                                                               |                           |                                                                                 |
| <code>max_rounds</code>                                                                                                                        | 2                         | Visual budget, separate from the fix budget                                     |
| <code>max_attempts</code> / <code>unavailable_policy</code>                                                                                    | 1 / warn                  | One request per comparison; outage warns                                        |
| <code>method</code>                                                                                                                            | <code>paired_vlm</code>   | Per-element pairing strategy                                                    |
| <code>reference</code>                                                                                                                         | <code>concept_crop</code> | Compared against the concept crop                                               |
| <code>crop_padding_px</code> / <code>min_element_px</code>                                                                                     | 6 / 16                    | Crop padding; smaller elements are skipped                                      |
| <i>Entity scale, framing, and UI proportion</i> ( <code>gates.scale</code> , <code>gates.subject_margin</code> , <code>gates.ui_scale</code> ) |                           |                                                                                 |
| <code>min_entity_px</code> / <code>max_entity_px</code>                                                                                        | 12 / 4096                 | Rendered content height bounds                                                  |
| <code>body_overflow_tolerance</code>                                                                                                           | 0.15                      | Collision body may exceed art by 15%                                            |
| <code>body_min_ratio</code>                                                                                                                    | 0.30                      | Collision body at least 30% of art                                              |
| <code>size_class_tolerance</code>                                                                                                              | 0.10                      | Slack on relative size ordering                                                 |
| <code>subject_margin</code> top, bottom, left, right                                                                                           | 0.05                      | Clear margin fraction per edge                                                  |
| <code>min_px</code> / <code>max_texture_px</code>                                                                                              | 8 / 4096                  | Rendered and source size bounds                                                 |
| <code>aspect_tolerance</code>                                                                                                                  | 0.18                      | Drift from the asset's own aspect ratio                                         |
| <code>max_canvas_fraction</code>                                                                                                               | 0.55 / 0.85 / 1.0         | Canvas share for <code>.hud</code> , <code>.panel</code> , <code>.banner</code> |
| <code>overlap_min_px</code>                                                                                                                    | 12                        | Overlap on both axes counting as collision                                      |
| <i>Scene layout</i> ( <code>gates.layout</code> )                                                                                              |                           |                                                                                 |
| <code>tolerance</code>                                                                                                                         | 0.04                      | Position drift as a fraction of canvas diagonal                                 |
| <code>min_coverage</code>                                                                                                                      | 1.0                       | All declared elements must render, by name                                      |
| <code>overlap_min_px</code> / <code>overlap_tolerance</code>                                                                                   | 12 / 0.12                 | Rendered-bounds collision                                                       |
| <code>offscreen_min_px</code>                                                                                                                  | 8                         | Permitted off-canvas overhang                                                   |
| <code>size_drift_tolerance</code>                                                                                                              | 0.02                      | Rendered size versus texture size                                               |
| <i>Interaction and anti-tunneling</i> ( <code>gates.interaction</code> , <code>gates.tunnel</code> )                                           |                           |                                                                                 |
| <code>min_hit_target_px</code>                                                                                                                 | 24                        | Measured hit body floor for pick targets                                        |
| <code>tunnel.assumed_fps</code>                                                                                                                | 60                        | Frame budget converting speed to travel                                         |
| <code>tunnel.speed_threshold</code>                                                                                                            | 200                       | Only faster bodies are checked                                                  |
| <code>tunnel.min_body_ratio</code>                                                                                                             | 1.0                       | Travel per frame stays under target body                                        |
| <i>Design scope caps</i> ( <code>design</code> )                                                                                               |                           |                                                                                 |
| <code>max_sprite_entities</code>                                                                                                               | 4                         | Entities the design may plan                                                    |
| <code>max_actions_per_entity</code>                                                                                                            | 3                         | Animated actions per entity                                                     |

| Key                                                             | Value     | Effect                                     |
|-----------------------------------------------------------------|-----------|--------------------------------------------|
| max_sprite_action_rows                                          | 15        | Total sprite-action rows                   |
| max_background_layers                                           | 4         | Background layers                          |
| max_prop_packs / max_ui_elements                                | 3 / 4     | Prop packs and distinct UI element rows    |
| max_concept_images                                              | 6         | Concept mockups, hard cap 8                |
| <i>Asset production and sanity (assets, gates.asset_sanity)</i> |           |                                            |
| image.max_retries                                               | 3         | Closed-loop first-frame retries            |
| image.retry_delay_seconds                                       | 10        | Backoff between attempts                   |
| image.safe_bg_color                                             | #FF00FF   | Chroma-key fill for mactable types         |
| seedance.max_quality_candidates                                 | 1         | One take per chain by default              |
| seedance.transient_max_attempts                                 | 3         | Retries on transient failure only          |
| matting.backend                                                 | auto      | Prefer remote matting, fall back on outage |
| sprite.fps / sprite.max_frames                                  | 12 / 24   | Spritesheet sampling                       |
| sprite.max_sheet_width                                          | 4096      | Texture-size limit                         |
| vlm.temperature                                                 | 0.0       | Deterministic judgment                     |
| vlm.slot_attempts_per_model                                     | 1         | One slot request, no retry                 |
| min_png_dimension_px                                            | 32        | Anti-placeholder per-axis floor            |
| min_unique_files / min_total_bytes                              | 3 / 50000 | Diversity and size floors                  |
| <i>Verification runtime (verify)</i>                            |           |                                            |
| play_max_wall_seconds                                           | 240       | Play-agent wall budget                     |
| record_duration_seconds                                         | 30        | Recording length                           |
| max_parallel_producers                                          | 5         | Concurrent producers                       |
| max_parallel_provider_producers                                 | 2         | Concurrent provider-backed producers       |
| skip_video                                                      | false     | Release includes recording unless waived   |

## Appendix C. Judgment Rubrics

Every rubric below is evaluated by the pinned vision-language model at temperature zero, and is condensed here from the validator adapters. Decomposing a judgment into named binary rubrics rather than a quality score follows the finding that instruction-guided multimodal judging is most usable when each question is narrow and explicit (Ku et al., 2024).

### C.1 Sprite First-Frame Rubrics

These run under `validate_image` before any animation is generated, because a defect in the first frame propagates into every frame built from it.

| ID | Name                 | Condition for passing                                                                                                    |
|----|----------------------|--------------------------------------------------------------------------------------------------------------------------|
| I1 | Identity match       | Species, gender, era, art style, and notable features match the description exactly; no substitutions or embellishments. |
| I2 | Hand-prop attachment | Every prop shows a visibly gripping hand or forearm; a prop merely near the body fails, however plausibly placed.        |
| I3 | Limb integrity       | All expected limbs present, torso-connected, at realistic joint angles; none missing, extra, duplicated, or clipping.    |
| I4 | Foot anchor          | Lowest visible point in the bottom 30% of the canvas; a floating or vertically centered figure fails.                    |
| I5 | In-frame             | Character and props unclipped, with at least 20% margin above the head and 15% each side for later motion.               |
| I6 | Background pure      | Uniform solid chroma-key color; no gradient, shading, noise, secondary object, or border.                                |
| I7 | Single character     | Exactly one full-body character; no duplicates, pose grids, UI overlays, captions, or frames.                            |
| I8 | View consistency     | View angle matches the request: side, three-quarter, front, or back.                                                     |

Table 14: Image rubrics I1–I8, applied to sprite first frames.

Projectile bodies use a separate set P1–P6, since a fireball has no feet (I4), hand (I2), or limbs (I3) to check. P1 identity, P5 background purity, and P6 single object mirror their character counterparts. P2 body integrity requires one coherent object and forbids a baked motion trail, trails being runtime particle effects rather than art. P3 centered with margin requires at least 8% clearance on every edge, unlike I4, where feet may rest on the bottom edge. P4 travel direction checks orientation against the described heading, passing as not applicable for symmetric bodies.

### C.2 Sprite Animation Rubrics

The animation rubrics run under `validate_asset_video` across two inputs: the raw video, where the solid background still makes margins measurable, and the matted frames that ship. A take ships only if it clears both groups. Projectile animations use PR1–PR6, replacing the two anatomy rubrics R4 and R5 with one body-integrity check.

| ID                                                | Name                  | Condition for passing                                                                                                                                    |
|---------------------------------------------------|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Framing group, judged on the raw video</i>     |                       |                                                                                                                                                          |
| R1                                                | In-frame              | Nothing is clipped in any frame, including peak motion: every swing, sweep, dash, and trail stays fully inside the canvas. Feet on the bottom edge pass. |
| R2                                                | Scale locked          | Size stays within roughly 10–15% of the first frame, judged late-against-first so creep is caught. Extending a limb passes; scaling the body does not.   |
| <i>Content group, judged on the matted output</i> |                       |                                                                                                                                                          |
| R3                                                | Identity preservation | Same colors, patterns, face, and proportions in every frame; no drift, swap, or morphing.                                                                |
| R4                                                | Hand-prop attachment  | Held items stay anchored to a hand or the body; nothing floats, detaches, or hovers without a visible grip.                                              |
| R5                                                | Limb integrity        | No limb disappears, duplicates, or bends impossibly; a limb erased by the matte fails exactly as one the model never drew.                               |
| R6                                                | Loop coherence        | Cyclic actions loop seamlessly, last frame matching the first; one-shot actions such as death or hit-react pass as not applicable.                       |
| R7                                                | Action fidelity       | The motion visibly performs the requested action; a static pose, wrong action, wrong direction, or scale drift alone fails.                              |

Table 15: Video rubrics R1–R7.

### C.3 Release Review

`analyze_video` scores a sealed recording of gameplay against configured floors: intention at 0.9, whether the video shows the game the design document describes; visual at 0.7; and interaction at 0.8, whether player action and game response are legible. Intention is the strictest floor in the system: a recording that does not show the intended game is a wrong-artifact problem, not a quality problem.

The critic then sees screenshots, that analysis, the play report, the verification report, the design document, and the quality rubric; it does not see source code, and its default position is that the build is not ready.

| Dimension                       | Scale, from 0.1 through 0.3 and 0.5 to 0.8+                                                                                |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <code>visual_quality</code>     | Incoherent; rough or accidental; intentional but uneven; profile-coherent, polished                                        |
| <code>gameplay_mechanics</code> | No gameplay evidence at all; partially working; mostly working; all designed mechanics visible                             |
| <code>completeness</code>       | Multiple major features missing; one or two missing; all present but rough; complete and refined                           |
| <code>polish</code>             | No transitions or feedback; basic transitions; has juice; feels polished                                                   |
| <code>asset_fitness</code>      | Missing, broken, or illegible; inconsistent or poorly integrated; runtime-fit but rough; cohesive and correctly integrated |

Table 16: Critic scoring dimensions and anchors.

The verdict rule is deliberately asymmetric: a build passes only when all five dimensions score 0.7 or above, is rejected outright when any falls below 0.5, and is rejected by default in between. This hedges against the verbosity and self-enhancement biases of model judges (Zheng et al., 2023). Three

overrides sit above the scores: a play agent's suspected bug caps gameplay mechanics at 0.3; weighted accuracy below 0.5 is a strong rejection signal; and any build-health failure—failing to boot, a fatal error, an empty screenshot—rejects outright before the scores are read.

## Appendix D. Agent Interface Specification

This is the reproducibility contract: what someone needs in order to re-implement any single agent and have it participate correctly. Phase order belongs to the orchestration skill and phase-internal craft to each agent specification, so neither is repeated here. Agents are dispatched with a flat set of **key: value** fields; field order carries no meaning and every path is absolute. Capability comes from the tool allowlist and the boundary hook rather than from the prompt, so a field naming a path the agent does not own is simply inert.

### D.1 Phase Interfaces

**Startup routing** runs in the orchestrator with no subagent. For the rest of the run it fixes the values of `genre`, `design_axes`, `interaction_profile`, `quality_profile`, and `level_model`, and it sets `level_design_required` exactly when the level model is authored. Fields that recur across interfaces are defined once in Table 18 and only named below.

Table 17: Phase-agent dispatch interfaces.

| Agent        | Inputs                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Returns                                                                                                                                                                                                                                                       |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| design       | <i>Required:</i> <code>game_idea</code> , <code>genre</code> , <code>design_axes</code> , <code>interaction_profile</code> , <code>quality_profile</code> , <code>level_model</code> , <code>game_dir</code> , <code>prompt_path</code><br><br><i>Writes:</i> the design document and multimodal evidence under <code>design/</code> .                                                                                                                                                              | <code>status</code> , <code>gdd_path</code> , <code>sections</code> , <code>scene_count</code> , <code>asset_count</code> , <code>concept_mockups</code> , <code>scene_coverage</code> , <code>ref_crops</code> , <code>warnings</code> , <code>issues</code> |
| level-design | <i>Dispatch:</i> authored levels only<br><i>Mode:</i> <code>create</code> , or <code>revise</code> with structured <code>issues</code><br><i>Required:</i> <code>gdd_path</code> , <code>genre</code> , <code>level_model</code> , <code>game_dir</code>                                                                                                                                                                                                                                            | <code>stage_count</code> , <code>level_files</code> , <code>has_boss</code> , <code>extensions</code> , <code>design_change_requests</code>                                                                                                                   |
| scaffold     | <i>Mode:</i> <code>scaffold</code> , or <code>reproject</code> with non-empty <code>changed</code><br><i>Required:</i> <code>gdd_path</code> , <code>game_dir</code> , <code>config_path</code> , <code>quality_profile</code> , <code>level_model</code><br><br>Reproject deletion is receipt-mediated: scaffold seals its own stale list, and the orchestrator can only consume that receipt, never supply or widen the paths.                                                                    | <code>contracts</code> , <code>files_created</code> , <code>stale_owned_paths</code> , <code>stale_receipt_id</code>                                                                                                                                          |
| implement    | <i>Dispatch:</i> one scene<br><i>Required:</i> <code>scene_name</code> , <code>gdd_path</code> , <code>game_dir</code> , <code>genre</code> , <code>interaction_profile</code> , <code>quality_profile</code> , <code>level_model</code> , <code>url</code><br><i>Optional:</i> <code>visual_evidence</code> , on visual fix rounds only<br><br><code>url</code> carries server identity for continuity; it does not authorize browser execution, and implement authors tests without running them. | <code>scene</code> , <code>files_changed</code> , <code>tests</code> , <code>issues</code> , <code>design_change_requests</code>                                                                                                                              |

| Agent             | Inputs                                                                                                                                                                                                                                                                                                                                                                                  | Returns                                                                                    |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| assets            | <i>Dispatch:</i> one batch<br><i>Mode:</i> generate or register-existing<br><i>Required:</i> gdd_path, genre, quality_profile, level_model, game_dir, and a batch scope<br><br>Workers write only owned files and sidecars, never the design document or the manifest.                                                                                                                  | assets_generated, assets_registered, sprites, maps, props, ui, metadata_sidecars           |
| verify            | <i>Mode:</i> iterate, full, or gate<br><i>Required:</i> game_dir, url, mode, quality_profile, level_model<br><i>Conditional:</i> gate for gate mode, step for step diagnostics, verify_run_id for every verify and final gate<br><br>Each canonical pass is exactly one run_verify_dag call. A changed list may drive a fast diagnostic but never substitutes for the unfiltered suite. | verify_run_id, producer checkpoints, tests, evidence paths, verify_report, pre_critic_gate |
| critic            | <i>Required:</i> game_dir, screenshots, video_analysis, play_report, verify_report, gdd_path, rubric_path, quality_profile<br><br>Receives no ref_dir and reads only the listed evidence. Its report must copy the verification run identity verbatim.                                                                                                                                  | verdict, top_issue, issue_category, scores                                                 |
| asset-integration | <i>Mode:</i> integrate<br><i>Required:</i> gdd_path, game_dir, genre, interaction_profile, quality_profile, level_model, url<br><i>Optional:</i> a scope of config, preloads, animations, visual, or verify<br><br>Dispatched only after both build branches join and the manifest is finalized; seals its receipt before any verification epoch opens.                                 | assets_wired, display_params, asset_integration_report                                     |

## D.2 Shared Field Dictionary

| Field               | Meaning                                                                           |
|---------------------|-----------------------------------------------------------------------------------|
| game_dir            | Absolute game-project root; MCP tools take the same path under the same name      |
| gdd_path            | Absolute path to the game design document                                         |
| genre               | Startup-selected genre; downstream agents never re-route it                       |
| quality_profile     | One of art_directed, responsive, pixel_art, minimal                               |
| level_model         | One of authored, procedural, board, none                                          |
| interaction_profile | Startup-selected physics and interaction route                                    |
| url                 | The exact local URL verified by the server launcher                               |
| ref_dir             | Absolute phase-specific reference directory; narrows the context an agent reads   |
| config_path         | Absolute path to gca.config.yaml                                                  |
| level_files         | Complete ordered array of absolute level paths; required only for authored levels |

Table 18: Fields shared across dispatch interfaces.

### D.3 Retry and Continuation

Internal retries belong to the producer or the MCP server and use configured bounds. The governing rule is that an unknown task state means wait, not retry: a timeout does not prove termination, and treating it as terminal is how duplicate work and split-brain state arise.

Recovery reopens the existing workspace in resume mode and reconciles durable run, build, verification, and in-flight task identities before dispatch continues; guarded state and audit updates recover valid pending transitions. Completed work is reused only when its identity and freshness still match, execution continues from the first incomplete or stale stage, and browser-backed work additionally requires a freshly validated endpoint. No partial write or artifact from an earlier run becomes current evidence.

| Phase                          | Continuation rule                                                                                                                                                                                                                           |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Design, level design, scaffold | No automatic retry. Design converts a bounded concept-generation failure into its documented warning fallback: complete layout and mechanics ship without the image.                                                                        |
| Implement                      | No automatic scene retry; partial issues enter the fix loop, and resume targets only incomplete scenes.                                                                                                                                     |
| Assets                         | Internal retries are bounded by configuration; a terminal partial may continue with only the missing keys.                                                                                                                                  |
| Asset integration              | One full dispatch first; scoped continuations are for confirmed timeouts only and run sequentially in the order config, preloads, animations, visual, verify.                                                                               |
| Verify                         | Resume the same epoch from checkpoints after a confirmed interruption.<br>Never restart completed nodes, and never open a fresh epoch for a transport failure. A new run starts only after the build changes or the previous one finalizes. |
| Critic                         | No automatic retry; a failed judgment blocks assessment rather than defaulting to either verdict.                                                                                                                                           |

Table 19: Per-phase retry and continuation policy.

### D.4 Return Envelope and Gate Interface

Every return carries a **status**. **done** means validate the outputs, persist completion, and continue; **partial** means retain the valid outputs, record every issue, and continue only when safe; **blocked** means report concrete issues and wait. Warnings are non-blocking, and issues describe incomplete or blocked work. The orchestrator never force-continues past a blocked return, never rewrites an agent’s return, and never treats an in-flight timeout as terminal. Each prohibition closes a route by which an orchestrator pressed to make progress could manufacture the appearance of completion, the exact failure the evidence architecture exists to prevent.

The orchestrator may call **gate\_check** directly, and **verify** in gate mode uses the same interface. Every response returns **blocked** plus a structured **failures[]** in which each record carries the failing **check**, routing hints naming the **axis** and owning **agent**, and a message; grouping by **agent** yields the parallel fixer dispatch described in Section 3. Each **step\_done** call writes its receipt to a step-numbered path, so one phase’s diagnostic cannot overwrite another’s. Three advisory diagnostics (**check\_ui\_contract**, **check\_td\_contract**, **check\_interaction\_contract**) expose deterministic contract gates for fast mid-build feedback without opening an epoch. They are explicitly not release authorities:

the shipping decision is `ready_to_ship` with a matching verification run identity, and the stop hook trusts only that gate's sealed result.

## Appendix E. Human Study Questionnaire

**Participant instructions.** Read the assigned prompt, then play Game A and Game B in the recorded order. System identities are hidden behind randomized labels. Rate each game from your own interaction and observation, without inferring quality from loading time outside the game, file names, or perceived authorship. Complete one copy for each of the 20 pairs in Table 4.

**Participant ID:** \_\_\_\_\_ **Session ID:** \_\_\_\_\_ **Date:** \_\_\_\_\_

**Report task no.:** \_\_\_\_\_ **Task/game name:** \_\_\_\_\_

**Pair position:** \_\_\_\_/20 **Randomization:** \_\_\_\_\_ **Order:** ☐ A-B ☐ B-A

**Shared response scale.** 1 = clearly not met; 2 = mostly not met; 3 = partly met; 4 = mostly met; 5 = fully met.

| Rating                   | Participant question                                                                                                        | Game A<br>(1–5) | Game B<br>(1–5) |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------|-----------------|-----------------|
| Requirement Satisfaction | How fully does the game realize the mechanics, content, progression, visible feedback, and outcomes required by the prompt? | _____           | _____           |
| Core Mechanics           | Do the controls, rules, state transitions, success/failure conditions, and restart behavior work during play?               | _____           | _____           |
| Content Depth            | Are the requested modes, levels, progression, challenge variation, and terminal content present?                            | _____           | _____           |
| Functional Visuals       | Can you see the important game state, HUD values, action feedback, and settlement information when needed?                  | _____           | _____           |
| Art & Presentation       | Beyond basic function, how coherent, readable, well composed, and visibly finished is the presentation?                     | _____           | _____           |

**Overall preference.** Considering prompt fulfillment, play, and presentation together:

☐ Game A ☐ Game B ☐ No preference

**Issue/comment.** Note any blocked action or concrete interaction/visual observation that influenced your ratings or preference.

---

---

Game A and Game B ratings are independent observations. Preference is categorical and is analyzed separately; it does not replace either five-item rating vector.