VGBENCH: CAN AGENT SCHEDULE A REAL-ENGINE GAME DEVELOPMENT?

Anonymous authors
Paper under double-blind review

ABSTRACT

Game generation is a challenging long-horizon task for modern coding agents. Existing game-generation benchmarks mostly measure the final artifact, leaving the development process unmeasured. In practice, the final artifact is only part of the result: an agent must also coordinate goals at different levels and produce useful intermediate builds. A sound process builds a playable core early, adds dependent features on top of it, and preserves earlier behavior as the game becomes more complex.

We introduce VGBench to evaluate this ability through two complementary settings. In the single-round setting, an agent receives a task with many requirements and a fixed development budget, while the benchmark embeds several intermediate checks; the score depends on the quality of the staged deliverables produced at these checkpoints. In the multi-round setting, the benchmark progressively adds requests and measures whether the agent can complete each new stage while maintaining the stability of existing functionality. Both modes use multiple rubrics for evaluation. An instrumentation agent prepares game states that are difficult to reach, and a player agent records the evidence used for scoring. VGBench contains 107 natural-language queries and nine stage increments, covering 17 game types and multiple harnesses. Evaluations of frontier coding agents show that end-to-end game generation remains challenging: the strongest multi-round configuration reaches 84.82 overall, while two thirds of the configurations remain below 72. Further analysis shows that agents with similar final game scores can differ sharply in their ability to deliver useful intermediate results, and this difference is reflected in the clarity and extensibility of the resulting games. VGBench provides a way to study long-horizon agents at the process level beyond final products.

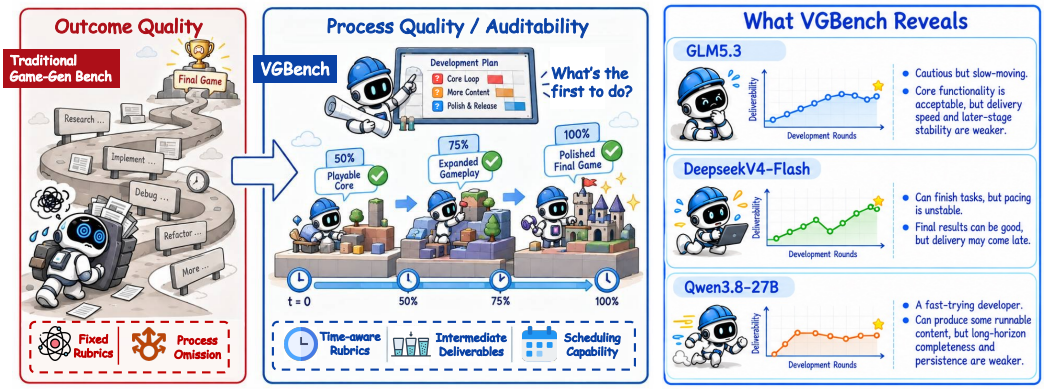


Figure 1: **(A):** Unlike most game-gen benchmarks that focus solely on runtime outcomes, VGBench additionally examines the alignment between the agent's running process and human preferences, i.e, whether it can establish a progressive deliverable intermediate artifacts. **(B):** Different agents exhibit intriguing contrasts and commonalities in their preferences for implementation on VGBench.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Large Language Models (LLMs) [14, 23] are increasingly acting as coding agents[17, 42, 3] that edit files and execute programs. Although modern coding agents have shown strong performance on a wide range of tasks, their ability in game development remains limited, owing to the substantial challenges of environment construction, exploration, and verification: an agent must first understand the intent and build a real, runnable game environment, then explore that environment with engine tooling and objectively judge the game's overall quality against expert criteria.

Recent game-oriented benchmarks and game-playing environments[21, 1, 29, 6, 37] and agentic software benchmarks[17, 42] have made important progress on this problem [2, 25, 5, 13, 15, 45, 46, 16]. However, the intermediate behavior of coding agents remains a black box whose relation to human preferences is an open question. Consider an industrial game development task that easily requires thousands of hours. To keep a project on schedule and to stop problems from compounding, human teams insert staged reviews and meetings: through periodic verification and artifact audition, they can discover the factors that would cause the project to drift and avoid them in advance. For a long-horizon agent deployed in practice, we likewise demand high deliverability at intermediate steps, which makes behavior auditing and ad-hoc human takeover or disaster-recovery design possible. Existing evaluations summarize a run by the artifact it produces at the end, so the builds that exist along the way are rarely scored [17, 42, 3, 26, 30].

For example, one agent may advance layer by layer, so that each intermediate artifact subsumes the previous one and remains self-contained. Another may finish individual sub-tasks without making the intermediate artifact usable as a whole, and assemble the pieces only at the end. The first process has lower error cost and produces a more stable, extensible project. In other words:

> Final-artifact quality is one criterion among several; how an agent runs and schedules the task matters in its own right.

In this case, we introduce VGBench, which measures whether an agent can imitate human development practices to deliver outcomes in an orderly manner. We term this capability "staged planning": while the agent is free to decide which requirements to implement first and when to demonstrate progress. VGBench encourages a progressive process that moves from core mechanics to presentational effects, To this end, it provides a large number of goals of mixed difficulty along with corresponding evaluation rubrics. These rubrics make latent judgments through two modes: ❶ **the single-turn mode**, which discloses the complete set of goals to the agent at the outset and embeds checkpoints at several intermediate time points, with these hidden checks strictly focusing on the progressive relationship of the game from core functionality to integrated capabilities; ❷ **The multi-turn mode**, which continuously introduces new requirements to examine whether the agent can improve game quality in a stable and non-regressive manner.

After each version submission, the evaluation enters the actually running game. VGBench provides fine-grained runtime control and evidence storage to enable in-depth inspection of game quality: a fixture mechanism bypasses states that would otherwise require lengthy play to reach, while an independent reviewer interacts with that version and records cited observations for each judgment. Through experiments on a wide range of frontier agents, we find that several agents performing well on final artifacts differ markedly in their staged delivery capability: two configurations whose single-round endpoints fall within 1.5 points of each other still differ by more than 35 points in the share of requirements delivered on time. On the other hand, interpretability experiments show that agents that complete development through layer-by-layer progression produce artifacts with far greater extensibility than those from agents that develop chaotically. These results suggest that current coding agents can often produce game-like software, but still struggle to reliably follow a decent development procedure all the way to an extensible artifact in a real engine. Our contribution can be summarized as:

- We propose VGBench that measures both requirement completion and process-level quality.
- We introduce two evaluation modes: single-round mode tests autonomous scheduling, and multi-round tests execution under an explicit development order.

Table 1: Comparison of VGBench with recent game-oriented benchmarks. ✓ full support, △ partial support, × absent. "E2E Gen." means generation from natural-language input, and "Inter.-build Test" means interactive testing of builds produced during development.

Benchmark	E2E Gen.	Real engine	White-box eval.	Black-box eval.	Fixture support	Multi-modal	Inter.-build Test
GameCraft-Bench [25]	✓	✓	△	✓	×	✓	×
GameGen-Verifier [15]	✓	△	✓	△	✓	✓	△
PlaytestArena [13]	✓	×	×	✓	×	✓	△
GameDevBench [5]	×	✓	✓	×	△	△	×
AutoUE [45]	✓	✓	△	✓	×	△	×
GameXpert-Bench [2]	✓	✓	△	✓	△	✓	×
V-GameGym [46]	✓	×	△	✓	×	✓	×
GBQA [16]	×	△	✓	✓	✓	△	×
VGBench	✓	✓	✓	✓	✓	✓	✓

- We build an evaluation workflow that connects intermediate game builds to evidence from actual play. This workflow supports analysis of staged delivery, functional regression, and the effect of stage-level guidance on long-horizon game development.

2 Related Work

Game generation and game benchmarks. Game generation emerged from procedural content generation, where levels and rules are produced by learned models [35, 34, 18, 31]. Mixed-initiative tools keep a human designer involved [22, 33, 36, 7]. A separate line builds environments and asks how well an agent plays an existing game [1, 29, 6, 10], from puzzle worlds to open-ended sandboxes [4, 20, 9, 8]. Playing a game and building one are different skills, and that line measures playing rather than building. Building a game has since become practical, so a growing set of benchmarks measures it [25, 45, 2, 5, 21] (Table 1). Others instead evaluate games that already exist, by playing them or by injecting runtime states [13, 15, 16, 46]. Both groups score only the finished artifact, so the development that produced it remains unobserved. VGBench instead reviews every intermediate build, so a requirement earns credit as soon as it becomes observable.

Long-horizon Agents. Long-horizon agents must maintain a useful state across many decisions, tool calls, and revisions. ReAct interleaves reasoning with actions, while Reflexion and Self-Refine use feedback to revise a plan or an intermediate result [43, 32, 28]. Tree of Thoughts and AgentBench study structured search and multi-step interaction, and Voyager examines sustained skill acquisition in an open-ended game world [44, 24, 37]. Software agents such as SWE-agent and OpenHands connect language models to repositories, terminals, and development tools [42, 38]; WebArena, OSWorld, VisualWebArena and similar benchmarks evaluate longer interaction chains in web, desktop, or embodied environments [47, 41, 19, 11, 27, 40]. Multi-agent frameworks such as AutoGen and MetaGPT distribute work across roles and conversations [39, 12]. These studies improve planning, feedback, and tool use, but their primary outcomes are task success, interaction success, or final software behavior. VGBench adds a different measurement target: it records successive executable game builds and scores when requirements become usable and whether they survive later changes.

3 VGBench Methodology

3.1 Benchmark Objective and Task Representation

Game development is a sequence of builds that depend on one another. A combat action depends on a state transition, feedback depends on that transition, and a higher-level design goal depends on several earlier systems. This dependency structure motivates our focus on the development trajectory.

At each observation, the project has a concrete executable state, requirements that are already observable, and requirements that remain to be built. VGBench asks whether an agent can move this

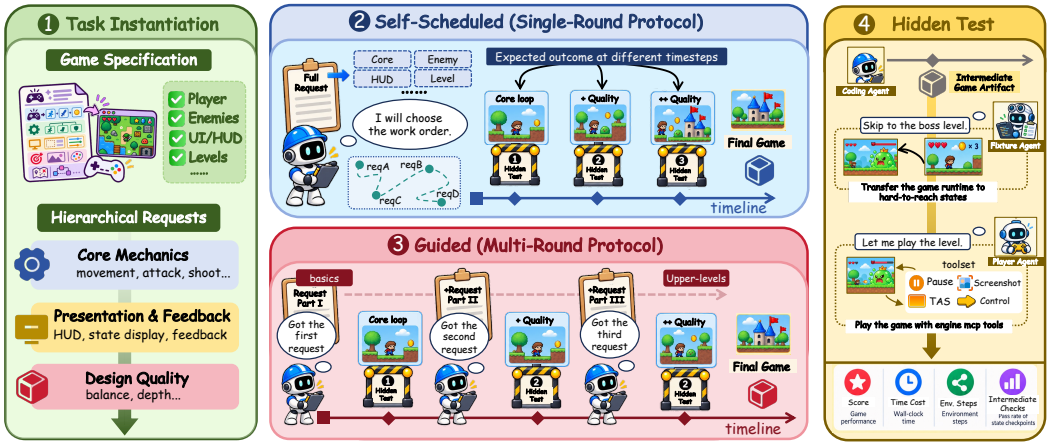


Figure 2: Overview of VGBench. A game-generation query is developed under a single-round or multi-round protocol, with hidden tests that measure intermediate deliverability. The final game is judged in a real engine by a fixture agent and a player agent.

state forward without losing useful work. For task i, let p_i be the game task, R_i the scored requirements, B_i the active development budget, and $\boldsymbol{d}_i$ the schedule of intermediate observations. The task collection is

$$\mathcal{D} = \{(p_i, R_i, B_i, \boldsymbol{d}_i)\}_{i=1}^{N}. \tag{1}$$

Each task contains a natural-language game brief and a rubric whose requirements can be judged in the running game. Requirements are written as observable outcomes, so the rubric is independent of implementation details and the agent remains free to organize its code. The rubric also records when a requirement becomes relevant and which earlier capabilities it depends on. These annotations support both evaluation modes with the same score definitions.

The scored requirements are divided into three groups, core mechanics, presentation, and design quality, defined in Section 3.4. The groups give a common language for asking which part of game development has become reliable at each point in the trajectory.

3.2 Single-Round Staged Delivery

Single-round evaluation gives the agent the complete game request at the start of one development session, from core mechanics to presentation and design goals. The agent chooses the implementation order, resolves dependencies, decides when to stop polishing a feature, and submits builds at the scheduled review points. A strong run should establish a playable core before adding disconnected effects.

At each scheduled point, we freeze the project and evaluate it as a game. The agent then continues from that same project. Each checkpoint is therefore a real development handoff: it records what a team could have played, inspected, and extended at that time.

Let $y_{rj} \in \{0, 1\}$ indicate whether requirement r is satisfied at observation j, let t_j be the active development time of that observation, and let d_r be the requirement's scheduled delivery time. Let F denote the final review. We define the delivery time of a requirement as $\tau_r = \min\{t_j : y_{rj} = 1\}$, with $\tau_r = +\infty$ when the requirement is never observed. The requirement contributes to the single-round score when it is present in the final build, and it receives the full delivery credit when $\tau_r \le d_r$:

$$q_r = y_{rF}\left[\alpha + (1-\alpha)\mathbb{I}\{\tau_r \le d_r\}\right], \tag{2}$$

where α controls the relative weight of final completion and timely delivery, and the benchmark applies the same rule to every task. This score separates two outcomes that are often collapsed in end-to-end evaluation: a game that reaches the target only at the deadline, and a game that reaches a usable target early and then extends it. A requirement that is completed early but later lost forfeits its delivery credit, because the final build must carry the development forward.

We summarize the trajectory with two measures. Timely delivery (TD) is the fraction of requirements usable at their scheduled delivery point. Regression rate (RegR) is the fraction of requirements

observed during the run that are absent from the final build. Let $\mathcal{R}_i^{\text{obs}} = \{r \in R_i : \tau_r < +\infty\}$ denote the requirements observed at least once. Then:

$$\text{TD}_i = \frac{1}{|R_i|} \sum_{r \in R_i} \mathbb{I}\{\tau_r \le d_r\} \quad \text{RegR}_i = \frac{1}{|\mathcal{R}_i^{\text{obs}}|} \sum_{r \in \mathcal{R}_i^{\text{obs}}} \mathbb{I}\{y_{rF} = 0\}. \tag{3}$$

TD falls when a requirement arrives after the point at which it could still have guided later work, and RegR rises when an earlier requirement is missing from the final build.

3.3 Multi-Round Guided Development

Multi-round evaluation turns the same game request into a sequence of development rounds. The first round asks for the smallest coherent playable version. Later rounds add presentation, interaction depth, progression, and design goals that depend on what came before. After every round, the agent receives the next request and continues from the submitted project. The game is reviewed at the end of each round, and the final review applies the complete rubric.

This setting tests whether an agent can extend an existing game without treating each round as a fresh generation task. Each new request should build on the earlier project, so the agent must choose representations, interfaces, and content that support later work. An agent that ignores this dependency may complete the new requirements while discarding earlier functionality.

Let $k(r)$ denote the round in which requirement r is introduced, and let y_{rj} indicate whether it passes at round j. Endpoint completion records whether the requirement is present in the final game. Preserved completion requires it to pass when introduced and at every later review:

$$e_r = y_{rK}, \qquad q_r^{\text{multi}} = \prod_{j=k(r)}^{K} y_{rj}. \tag{4}$$

We report e_r and q_r^{multi} separately. Their difference identifies requirements that were available at one point but disappeared later. Let $R_k = \{r : k(r) = k\}$ be the requirements introduced at round k, and let $P_k = \{r : k(r) < k,\ y_{r,k-1} = 1\}$ be the requirements that were complete when round k began. New-stage completion (NSC) measures how many new requirements pass in that round. Retention rate (RetR) measures how many previously complete requirements remain valid:

$$\text{NSC}_k = \frac{1}{|R_k|} \sum_{r \in R_k} y_{rk}, \qquad \text{RetR}_k = 1 - \frac{1}{|P_k|} \sum_{r \in P_k} \mathbb{I}\{y_{rk} = 0\}, \tag{5}$$

The indicator in RetR_k counts requirements that were complete before round k but fail during it. If a round adds two requirements and both pass, $\text{NSC}_k = 1$; if it also breaks one of five earlier requirements, $\text{RetR}_k = 4/5$. Doing nothing scores 0 on NSC and 1 on RetR. The measures are complementary: NSC records new progress, while RetR records what that progress preserves.

3.4 Rubrics and Play-tests

VGBench evaluates each game at three levels: core mechanics, presentation, and design quality. Each level asks a different question about the same playable artifact:

- *Core mechanics* asks whether the game works as a game. It covers the input and state-transition loop (movement, interaction, combat, or other task-specific actions) and the rules that connect these events. A requirement belongs here when it is necessary for play.
- *Presentation* asks whether the game communicates what is happening. It covers interface information, animation, effects, sound, camera response, hit or damage feedback, transitions, and other signals that connect a player's action to the resulting state. A mechanic can be implemented but still fail this dimension when the player cannot read its result.
- *Design quality* asks whether the systems form a coherent experience. It covers level or encounter structure, pacing, challenge, balance, variety, progression, goal clarity, and the use of mechanics over time. These requirements are evaluated through a situation or play sequence, not by inspecting an isolated source-level feature.

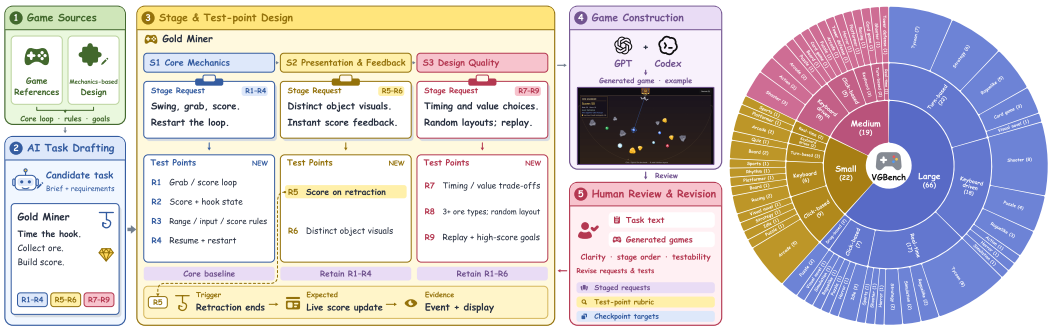


Figure 3: VGBench task construction. We collect game-generation tasks, expand their attributes into requirements at three levels, and have human experts check difficulty, testability, and clarity in the real engine. **Right:** Task distribution by game scale and gameplay type.

For task i, the requirement set is partitioned as $R_i = R_i^{\text{core}} \dot{\cup} R_i^{\text{presentation}} \dot{\cup} R_i^{\text{design}}$. For a requirement group g, its task-level score is the mean of its requirement scores:

$$Q_i^g = \frac{1}{|R_i^g|} \sum_{r \in R_i^g} q_{ir}, \qquad g \in \{\text{core}, \text{presentation}, \text{design}\}. \tag{6}$$

The group scores show where a trajectory gains or loses quality. A high core score with a low presentation score describes a functional but unreadable game. A high presentation score with a low design score describes a polished surface without a sustained experience.

Every requirement is judged through a concrete runtime observation. The evaluator selects or constructs a state in which the behavior can be tested. A fixture may place the player near an encounter, provide a required inventory or progression state, or restore a configuration that would take too long to reach through ordinary play. It only shortens the path to the test; the verdict rests on what the player observes. The player then interacts with the running game from that state.

3.5 Benchmark Construction Pipeline

Figure 3 (left) shows the construction pipeline. We collect tasks from existing benchmarks such as GameCraft-Bench, human-authored briefs, and well-known open-source games. For each game, we write an attribute report covering its mechanics, target experience, and art style. An LLM expands this report into concrete requirements at the three levels defined in Section 3.4.

Human experts then judge each requirement for difficulty, testability, and clarity in the real engine. Requirements that fail are revised or removed, and the same review assigns the difficulty strata reported in Table 5. Figure 3 (right) summarizes the benchmark by game scale and gameplay type.

4 Experiment

4.1 Experiment Setup

Experiment setup. The evaluation uses 107 natural-language game requests spanning 17 game types and nine staged increments. Each request specifies a complete game, a requirement rubric, and the intermediate goals used by the multi-round condition. The single-round and multi-round runs start from the same project state and use the same final rubric; they differ only in the information available during development. We evaluate GLM5.3, DeepseekV4-Flash, Minimax-M3, Qwen3.8-27B, GPT-5.6-Sol, and Claudecode4.8 with Codex, Claude Code, and DeepSeek Harness, forming the 21 rows in the comparison. Every configuration receives the same task files, initial project, active development budget, checkpoint schedule, and runtime review protocol.

To review a build we play it. We reach the behavior under test by ordinary play whenever practical, and otherwise start the reviewer near the relevant state; in both cases the reviewer performs the interaction and records what happens.

Table 2: VGBench single-round results. Quality reports endpoint scores. Process reports timely delivery (TD) and regression rate (RegR).

Base model	Harness	Overall	Core	Presentation	Design	TD	RegR
GLM5.3	Codex	70.81	72.08	74.45	75.17	68.29	8.63
	Claude Code	82.03	82.66	89.60	89.62	78.11	5.10
	DeepSeek Harness	78.92	81.30	75.88	78.59	92.47	1.84
DeepseekV4-Flash	Codex	60.86	61.78	59.51	60.75	38.10	16.00
	Claude Code	77.54	78.56	78.27	75.44	56.33	10.11
	DeepSeek Harness	66.06	67.39	66.08	64.25	33.10	16.34
Minimax-M3	Codex	51.27	48.63	55.42	47.18	64.71	17.84
	Claude Code	53.06	57.31	49.87	52.09	69.42	13.68
	DeepSeek Harness	50.39	56.48	42.26	52.13	58.36	22.91
Qwen3.8-27B	Codex	46.18	41.72	49.36	47.45	68.37	11.26
	Claude Code	47.92	52.08	44.67	46.91	65.14	15.73
	DeepSeek Harness	52.76	55.18	49.83	52.41	62.58	17.46
GPT-5.6-Sol	Codex	73.25	78.73	71.62	66.96	72.80	1.14
	Claude Code	63.01	66.42	61.97	59.22	50.72	1.88
	DeepSeek Harness	64.22	69.27	62.49	58.78	69.12	1.92
Claudecode4.8	Codex	71.34	75.58	68.27	68.02	44.29	3.23
	Claude Code	83.79	89.78	80.67	78.74	80.64	2.89
	DeepSeek Harness	73.31	77.48	71.65	69.28	73.89	4.69

Table 3: VGBench multi-round results. Quality reports endpoint scores. Process reports new-stage completion (NSC) and retention rate (RetR).

Base model	Harness	Overall	Core	Presentation	Design	NSC	RetR
GLM5.3	Codex	65.91	66.52	75.54	74.25	70.09	13.38
	Claude Code	83.18	83.43	85.96	81.35	85.83	19.03
	DeepSeek Harness	74.75	78.55	77.48	67.41	88.63	31.27
DeepseekV4-Flash	Codex	61.13	59.44	61.49	63.16	83.33	96.43
	Claude Code	64.81	67.12	63.26	62.90	86.27	100.00
	DeepSeek Harness	81.63	84.77	79.73	78.66	92.77	89.00
Minimax-M3	Codex	28.96	36.77	24.83	27.41	61.08	79.52
	Claude Code	33.14	29.76	35.28	26.37	52.83	81.67
	DeepSeek Harness	20.47	24.92	15.38	18.74	66.19	82.85
Qwen3.8-27B	Codex	26.41	30.18	21.74	24.09	59.83	81.46
	Claude Code	23.68	27.56	20.31	18.44	55.27	78.19
	DeepSeek Harness	21.87	22.64	20.93	21.74	48.36	73.82
GPT-5.6-Sol	Codex	77.78	76.92	91.67	69.44	84.62	100.00
	Claude Code	68.40	73.06	67.65	62.67	83.33	98.68
	DeepSeek Harness	68.14	71.73	68.69	63.16	84.92	98.73
Claudecode4.8	Codex	82.71	83.63	82.91	80.61	96.03	89.96
	Claude Code	84.82	92.64	79.87	78.31	97.41	97.67
	DeepSeek Harness	70.04	75.02	64.07	68.10	90.99	98.68

4.2 Main Results

Judge-human agreement. We first verify that the automated judge agrees with human scoring. Across 303 review rounds the human and judge means are 2.42 and 2.45, the per-stage means differ by at most 0.07, and the two orderings agree at $r = 0.870$ Pearson and $\rho = 0.791$ Spearman (both $p < .001$). Table 4 lists the per-stage values and Figure 4 shows the association curve. Agreement is closest on core mechanics, where both means are 2.41.

Single-round results. Table 2 separates endpoint quality from process quality. Model choice has a larger effect on the endpoint than harness choice: the average endpoint scores range from 48.95 to 77.25 across base models, compared with a 5.6-point spread across harnesses. The exception is GPT-5.6-Sol, which reaches 73.25 with Codex but 63.01 with Claude Code. Process measures provide a different ordering. DeepseekV4-Flash with Claude Code scores 77.54 overall but delivers only 56.33% of requirements on time, whereas GLM5.3 with DeepSeek Harness scores 78.92 and reaches 92.47% timely delivery. Regression is lowest for GPT-5.6-Sol (1.14–1.92) and Claudecode4.8 (2.89–4.69), while Minimax-M3, Qwen3.8-27B, and DeepseekV4-Flash exceed 10 on at least two harnesses. Thus, a high endpoint score is usually accompanied by lower regression, but it does not determine when the work becomes usable.

Multi-round results and configuration profiles. The results are listed in Table 3. Staging changes the ranking because every round adds new work while requiring the previous game to remain intact. Claudecode4.8 and GLM5.3 retain the top two positions. GPT-5.6-Sol improves from 66.83 to 71.44 on average and passes DeepseekV4-Flash, while Minimax-M3 and Qwen3.8-27B lose 24.05 and 24.97 points, respectively. New-stage completion and retention measure different abilities: their correlation is only 0.15 across the 18 rows, and GLM5.3 with Claude Code reaches

Table 4: Judge-human calibration (n = review rounds; every review round of every stage's evaluation is one sample). Stage 1 covers core mechanics, Stage 2 polish, and Stage 3 content and design. r is Pearson and ρ the Spearman rank correlation between judge and human scores across all n samples.

	Stage 1	Stage 2	Stage 3	Overall
n	104	99	100	303
Human	2.41	2.37	2.47	2.42
Judge	2.41	2.44	2.49	2.45
Δ	+0.00	+0.07	+0.03	+0.03

$r = 0.870, \quad \rho = 0.791 \quad (p < .001)$

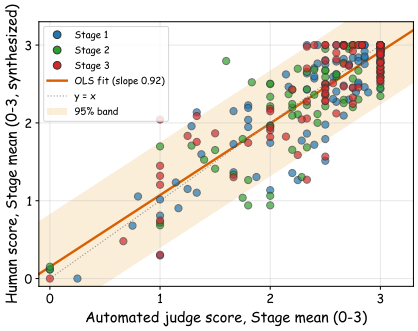


Figure 4: Association curve between LLM evaluation and human-perceived quality (300+ samples).

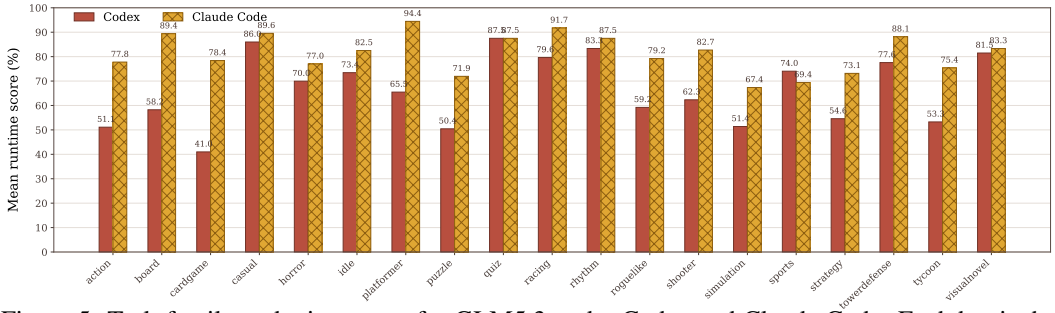


Figure 5: Task-family endpoint scores for GLM5.3 under Codex and Claude Code. Each bar is the mean requirement score within one family, and the labels on top give the two values.

85.83% new-stage completion but retains 19.03% of earlier requirements, while GPT-5.6-Sol and DeepseekV4-Flash retain 98–100% on most rows. The process columns therefore give several configurations distinct behavioral profiles. GLM5.3 with DeepSeek Harness is an early-delivery configuration, reaching 92.47% timely delivery with 1.84% regression in single-round runs, but its multi-round endpoint is 74.75 and its design score is 67.41. GLM5.3 with Claude Code is more balanced, reaching 82.03 in single-round quality and 83.18 in multi-round quality. Claudecode4.8 with Claude Code combines high endpoint quality with low single-round regression (2.89%) and high multi-round completion and retention (97.41% and 97.67%). GPT-5.6-Sol is similarly stable, while Minimax-M3 and Qwen3.8-27B can produce partial single-round builds but fall to 20–33 and 22–26 in multi-round quality. The same protocol thus helps agents that can schedule and maintain a project and exposes agents better suited to shorter or more local changes.

4.3 Behavioral Analysis

The endpoint score says how much of the requested game exists. The process measures explain how that result was produced. Core mechanics ask whether the game works, presentation asks whether the player can understand it, and design quality asks whether the systems hold together over a session. Across the 18 single-round rows, these groups average 67.36, 63.99, and 63.50, respectively; core is the highest group on 14 rows. The ordering follows the artifact's dependency structure: agents usually make the loop run before they make it readable, and make both before coordinating several systems over a full session.

Task families (Figure 5) show where this general pattern changes. Under Claude Code, ten families remain above 80, while puzzle, simulation, sports, strategy, and tycoon stay below 76. The largest harness gaps occur in platformer (65.5 with Codex versus 94.4 with Claude Code), board (58.2 versus 89.4), and cardgame (41.4 versus 78.4); Codex leads only on sports (74.0 versus 69.4). Families built around one interaction loop transfer more reliably across harnesses, whereas families that require several systems to remain consistent are more sensitive to the development interface.

Difficulty determines how strongly these dependencies affect the result (Table 5). Single-round scores are stable on easy and medium tasks (78.33 and 79.35) but fall to 63.37 on hard tasks. Multi-

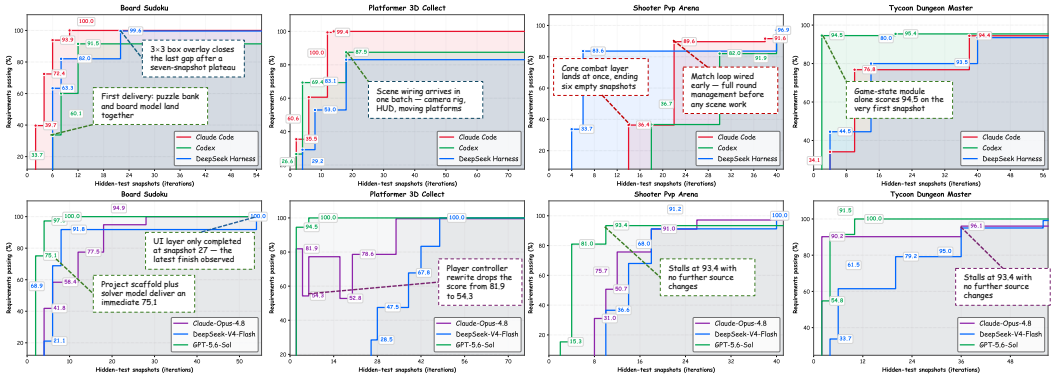


Figure 6: Hidden-test ladders. Each panel follows one game under one configuration, read as the share of task requirements satisfied at each hidden-test snapshot. The top row varies the harness and the bottom row the base model; callouts mark turning points.

Table 5: Difficulty-stratified endpoint performance. Tasks are grouped before evaluation by requirement count, number of staged increments, and the dependency structure among requirements. The final two rows report endpoint scores for the two protocols.

	Easy	Medium	Hard
# tasks	21	41	45
# requirements	7.43	12.37	17.71
Single-round endpoint score	78.33	79.35	63.37
Multi-round endpoint score	76.80	65.45	56.63

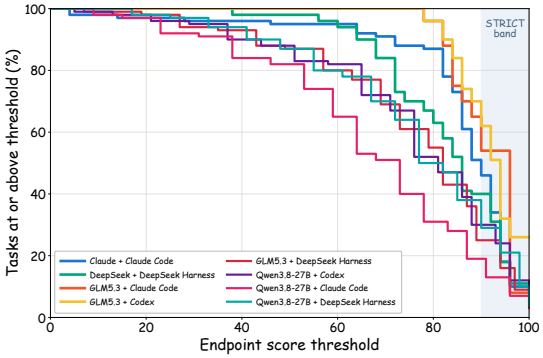


Figure 7: Endpoint-score survival comparison. Each curve shows the fraction of one config's scoring above the threshold on the horizontal axis.

round scores decline at every level, from 76.80 to 65.45 to 56.63. The gap between modes is only 1.53 points on easy tasks, but 13.90 on medium tasks and 6.74 on hard tasks, so explicit staging is nearly free for small games and most consequential when several systems must be extended together. The ladder panels (Figure 6) show the same effect in time: steady staircases indicate incremental delivery, late jumps indicate batch work, and drops after a flat stretch indicate regression. In one Platformer 3D Collect run, a controller rewrite lowered the score from 81.9 to 54.3. These trajectory shapes explain why configurations with similar endpoints can have different process scores.

Runtime Evidence and Case Studies Figure 7 shows survival curves, i.e., the score distribution across configurations. Most configurations keep their tasks above the low-score range; the main separation appears in the upper tail. This is why a configuration can have a moderate mean yet retain many strong tasks, or a higher mean with a longer tail of partial builds. The strict band makes that distinction visible. The appendix grounds this distributional view in runtime evidence: DeepseekV4-Flash delivers 33.10% of requirements on time and loses 16.34% of observed work, GLM5.3 with Codex reaches 68.29% timely delivery with 8.63% regression, and Claudecode4.8 with Claude Code reaches 80.64% timely delivery with 2.89% regression. These cases show how late delivery, stable extension, and regression appear in the executable artifact.

5 Conclusion

We introduce VGBench, which evaluates game generation as a sequence of executable builds. It measures the endpoint game, when requirements become usable, and whether earlier behavior survives later changes. The single-round protocol tests scheduling under a complete request; the multi-round protocol tests continuation under an explicit sequence of additions. VGBench introduces a new way to evaluate the process-level quality of long-horizon agentic tasks.

AI USE STATEMENT

In this work, we used generative AI tools to edit the manuscript for readability, grammar, and presentation. We did not use generative AI tools for the required research tasks listed in the ICLR 2027 AI Policy for Authors; all other such tasks are not applicable to this work. The authors reviewed all AI-assisted text and take responsibility for the final content of this paper, including its text, claims, and artifacts.

REFERENCES

[1] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[2] Kun Chen, Haorong Hong, Peizhong Gao, Jianfeng Lin, Tongxu Luo, Yuxuan Xie, Chenxu Liu, Jieling He, Zhongyuan Liu, and Zeno Zeng. GameXpert-Bench: How far are coding agents from expert game development? *arXiv preprint arXiv:2608.21833*, 2026. URL https://arxiv.org/abs/2608.21833.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[4] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: First steps towards grounded language learning with a human in the loop. In *ICLR*, 2019.

[5] Wayne Chi, Yixiong Fang, Arnav Yayavaram, Siddharth Yayavaram, Seth Karten, Qiuhong Anna Wei, Runkun Chen, Alexander Wang, Valerie Chen, Ameet Talwalkar, and Chris Donahue. GameDevBench: Evaluating agentic capabilities through game development. *arXiv preprint arXiv:2602.11103*, 2026. URL https://arxiv.org/abs/2602.11103.

[6] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *ICML*, 2020.

[7] Michael Cook and Simon Colton. Ludus ex machina: Building a 3d game designer that competes alongside humans. In *International Conference on Computational Creativity*, 2013.

[8] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *NeurIPS*, 2022.

[9] William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. In *IJCAI*, 2019.

[10] Danijar Hafner. Benchmarking the spectrum of agent capabilities. In *ICLR*, 2022.

[11] 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. *arXiv preprint arXiv:2401.13919*, 2024.

[12] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, and Chenglin Wu. Metagpt: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

[13] Yixu Huang, Bo Li, Na Li, Zhe Wang, Kaijie Chen, Haonan Ge, Qingyi Si, Yuanzhe Shen, Ruihan Yang, Guangjing Wang, and Hongcheng Guo. GUI agents for continual game generation. *arXiv preprint arXiv:2605.28258*, 2026. URL https://arxiv.org/abs/2605.28258.

[14] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[15] Chaobo Jia, Ruipeng Wan, Ting Sun, Weihao Tan, Borui Wan, Yuxuan Tong, Guangming Sheng, and Hong Xu. GameGen-Verifier: Parallel keypoint-based verification for LLM-generated games via runtime state injection. *arXiv preprint arXiv:2605.07442*, 2026. URL https://arxiv.org/abs/2605.07442.

[16] Shufan Jiang, Chios Chen, and Zhiyang Chen. GBQA: A game benchmark for evaluating LLMs as quality assurance engineers. *arXiv preprint arXiv:2604.02648*, 2026. URL https://arxiv.org/abs/2604.02648.

[17] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? *CoRR*, abs/2310.06770, 2023. URL https://arxiv.org/abs/2310.06770.

[18] Ahmed Khalifa, Philip Bontrager, Sam Earle, and Julian Togelius. Pcgrl: Procedural content generation via reinforcement learning. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2020.

[19] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *ACL*, 2024.

[20] Heinrich Kuttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. In *NeurIPS*, 2020.

[21] Brian La, Sejoon Chang, Ben Kim, Junyoung Bae, Aamish Ahmad Beg, Sei Chang, Gonzalo Gonzalez-Pumariega, and Kanav Goyal. GameEngineBench: Evaluating coding agents on real C++ runtime environments. *arXiv preprint arXiv:2607.03525*, 2026. URL https://arxiv.org/abs/2607.03525.

[22] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Mixed-initiative content creation. *Springer Handbook of Digital Games and Entertainment Technologies*, 2016.

[23] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[24] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating large language models as agents. *CoRR*, abs/2308.03688, 2023. URL https://arxiv.org/abs/2308.03688.

[25] Tongxu Luo, Rongsheng Wang, Jiaxi Bi, Chenming Xu, Zhengyang Tang, Jianlong Chen, Juhao Liang, Ke Ji, Shuqi Guo, Yuhao Du, Fan Bu, Wenyu Du, Xiaotong Zhang, Kyle Li, Shaobo Wang, Linfeng Zhang, Yuxuan Liu, Xin Lai, Chenxin Li, Yiduo Guo, Zhexin Zhang, Xinyuan Wang, Tianyi Bai, Ziniu Li, and Benyou Wang. GameCraft-Bench: Can agents build playable games end-to-end in a real game engine? *arXiv preprint arXiv:2606.17861*, 2026. URL https://arxiv.org/abs/2606.17861.

[26] Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. *arXiv preprint arXiv:2504.08942*, 2025. URL https://arxiv.org/abs/2504.08942.

[27] Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *CoRR*, abs/2401.13178, 2024. URL https://arxiv.org/abs/2401.13178.

[28] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, 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. *CoRR*, abs/2303.17651, 2023. URL https://arxiv.org/abs/2303.17651.

[29] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M. Lucas, Adrien Couëtoux, Jerry Liu, Jens M. Andersen, and Tommy Thompson. The 2014 general video game playing competition. In *IEEE Transactions on Computational Intelligence and AI in Games*, 2016.

[30] Andrey Podivilov, Vadim Lomshakov, Sergey Savin, Matvei Startsev, Roman Pozharskiy, Maksim Parshin, and Sergey Nikolenko. Agentlens: Production-assessed trajectory reviews for coding agent evaluation. *arXiv preprint arXiv:2607.06624*, 2026. URL https://arxiv.org/abs/2607.06624.

[31] Anurag Sarkar, Matthew Guzdial, Sam Snodgrass, Adam Summerville, Tiago Machado, and Gillian Smith. Procedural content generation via knowledge transformation (pcg-kt). *arXiv preprint arXiv:2305.00644*, 2023. URL https://arxiv.org/abs/2305.00644.

[32] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik R. Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *CoRR*, abs/2303.11366, 2023. URL https://arxiv.org/abs/2303.11366.

[33] Gillian Smith, Jim Whitehead, and Michael Mateas. Tanagra: Reactive planning and constraint solving for mixed-initiative level design. In *IEEE Transactions on Computational Intelligence and AI in Games*, 2011.

[34] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural content generation via machine learning. *IEEE Transactions on Games*, 10(3):257–270, 2018.

[35] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. *Search-Based Procedural Content Generation*. Springer, 2011.

[36] Mike Treanor, Bryan Blackford, Michael Mateas, and Ian Bogost. Game-o-matic: Generating videogames that represent ideas. In *FDG*, 2012.

[37] 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. *CoRR*, abs/2305.16291, 2023. URL https://arxiv.org/abs/2305.16291.

[38] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Minghao Liu, Mingming Zhang, Z. Z., et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

[39] 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. *arXiv preprint arXiv:2308.08155*, 2023.

[40] Zhiheng Xi, Dingwen Yang, Jiaqi Liu, Jixuan Huang, Honglin Guo, Baodai Huang, Tinggang Chen, Qi Zhang, Zhonghang Lu, Chenyu Liu, et al. Agentgym2: Benchmarking large language model agents in de-idealized real-world environments. In *ACL*, 2026.

[41] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *CoRR*, abs/2404.07972, 2024. URL https://arxiv.org/abs/2404.07972.

[42] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Kexin Pei, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.

[43] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *CoRR*, abs/2210.03629, 2022. URL https://arxiv.org/abs/2210.03629.

[44] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *CoRR*, abs/2305.10601, 2023. URL https://arxiv.org/abs/2305.10601.

[45] Lei Yin, Wentao Cheng, Zhida Qin, Tianyu Huang, Yidong Li, and Gangyi Ding. AutoUE: Automated generation of 3d games in unreal engine via multi-agent systems. *arXiv preprint arXiv:2603.07106*, 2026. URL https://arxiv.org/abs/2603.07106.

[46] Wei Zhang, Jack Yang, Renshuai Tao, Lingzheng Chai, Shawn Guo, Jiajun Wu, Xiaoming Chen, Ganqu Cui, Ning Ding, Xander Xu, Hu Wei, and Bowen Zhou. V-GameGym: Visual game generation for code large language models. *arXiv preprint arXiv:2509.20136*, 2025. URL https://arxiv.org/abs/2509.20136.

[47] 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. *CoRR*, abs/2307.13854, 2023. URL https://arxiv.org/abs/2307.13854.

A Implementation Details

Task Files. A VGBench task file contains the complete game specification, the scored requirement set, the stage annotations, and the review instructions. The specification describes the intended player experience. Each requirement translates one part of that experience into an observable acceptance condition. Stage annotations record the role of the requirement in the development sequence and the earlier capabilities on which it depends. Multi-round prompts reveal these annotations in order; single-round prompts retain them as part of the hidden evaluation schedule.

Budget and Checkpoints. The protocol fixes the active development budget, checkpoint times, stage budgets, and review limits. Single-round evaluation collects the builds submitted at the scheduled checkpoints and at the final deadline. Multi-round evaluation collects one build at the end of each stage and a final build after the last request. Every submitted build is archived before review, together with the active development time and the task state from which it was produced.

Rubric Examples. A movement requirement is satisfied when an input changes the player state in the intended direction and the result is visible in the running game. A restart requirement is satisfied when a completed or failed session can return to a fresh playable state. A display requirement is checked after the relevant state change, so the evidence includes both the displayed information and the event that should update it. A balance requirement uses a task-specific scenario in which the intended challenge or progression can be observed.

Instrumentation and Player Roles. Some requirements depend on states that are expensive to reach through ordinary play. The evaluator therefore prepares controlled starting states for review. The player reviewer begins from the assigned state, performs the required interaction, and records the observed result. This arrangement shortens the path to the relevant behavior while keeping the judgment grounded in the executable game. The run record links every verdict to its build, starting state, action sequence, and evidence.

Interactive Review Conditions. Runtime review uses the interaction controls available for the evaluated game. The reviewer records the initial state, action sequence, observed state, and evidence associated with each requirement. The protocol is defined by observable play, so the same rubric can be applied to projects with different internal structures while preserving the acceptance condition.

Archived Artifacts. Each run archives the task input, model and harness configuration, submitted builds, checkpoint timestamps, review observations, requirement judgments, and final report. The archive preserves the path from a requirement to the build and observation that support its score. It also allows the single-round and multi-round records for the same task to be paired after evaluation.

Recorded Run Fields. Each task record contains the task identifier, game type, model, harness, evaluation mode, active budget, checkpoint index, elapsed development time, submitted project state, runtime starting state, action sequence, evidence references, requirement verdicts, stage labels, delivery status, regression status, final score, stage scores, retention, and failure categories. Aggregation is performed from these task-level records. Threshold curves sweep a score threshold and count the fraction of tasks whose score remains above it.

Metric Definitions. Each evaluation mode reports four endpoint scores followed by two process measures, all computed from the requirement verdicts of a single run. The measures are defined formally in Section 3.2 and Section 3.3; we restate them here in words. *Timely delivery* (TD) is the fraction of requirements that became usable at the point at which they could still guide later work; a requirement that appears only after that point is counted as late. *Regression rate* (RegR) is the fraction of requirements observed at any point during development that are absent from the final build. *New-stage completion* (NSC) is the fraction of requirements introduced at the current round that are completed by the end of that round. *Retention rate* (RetR) is the fraction of requirements completed before a round that remain valid after it.

B RUNTIME CASE STUDIES

This appendix collects the runtime evidence in VGBench. Each figure covers one model–harness configuration and eight game tasks. Every task is shown with the requirement verdicts and the mean rubric score recorded during review, together with frames that illustrate the state on which the verdict rests. Figures 8–11 use recaptures of the archived build; a recapture illustrates the reviewed state and is neither an original review frame nor a new evaluation. Figures 12 and 13 use the original review screenshots.

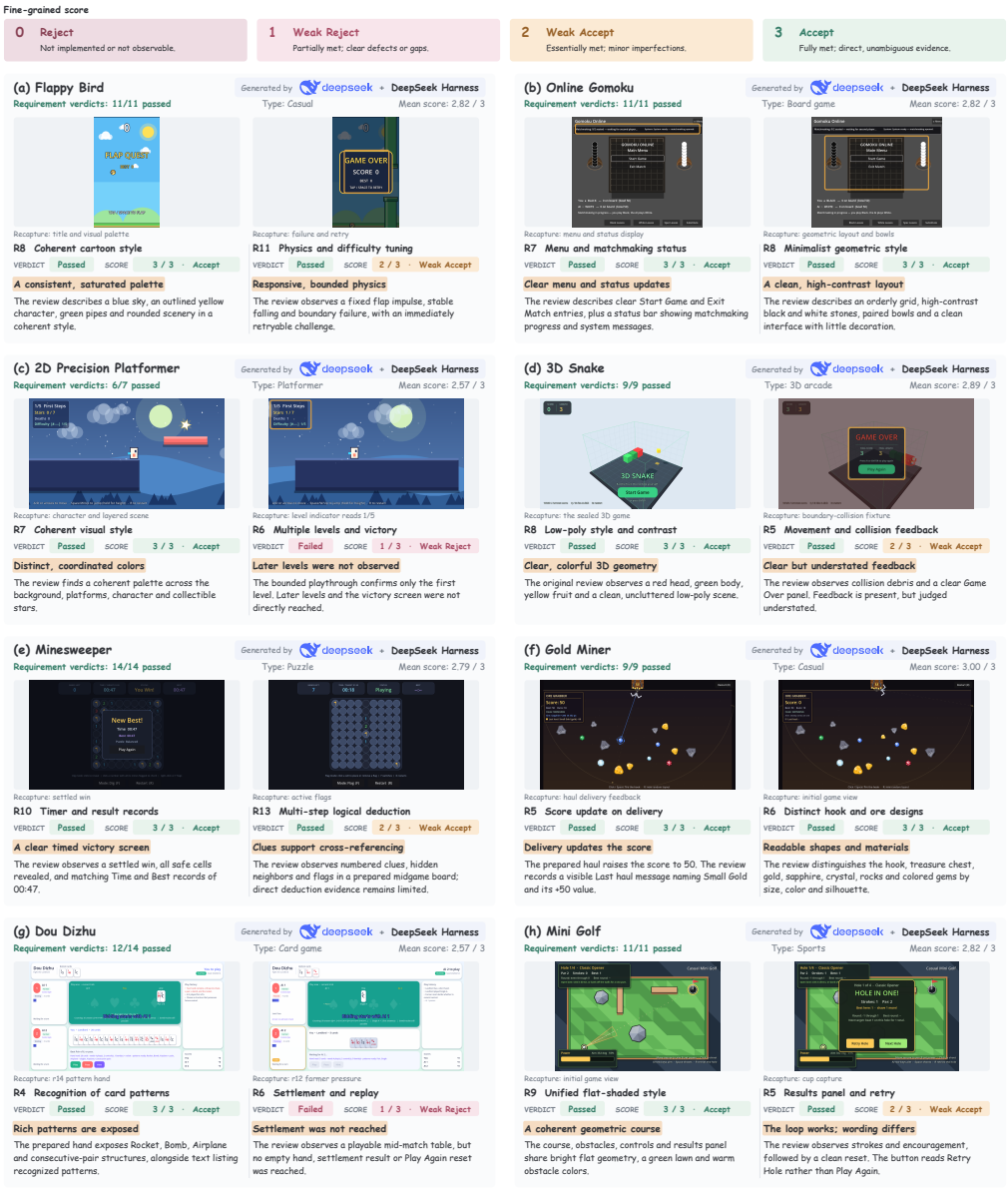


Figure 8: **DeepSeek with the DeepSeek Harness, part 1.** Eight single-round tasks: Flappy Bird, Online Gomoku, 2D Precision Platformer, 3D Snake, Minesweeper, Gold Miner, Dou Dizhu and Mini Golf. Each task reports its requirement verdicts and the mean rubric score recorded during review, with frames that illustrate the state behind the verdict. The frames are recaptures of the archived build.

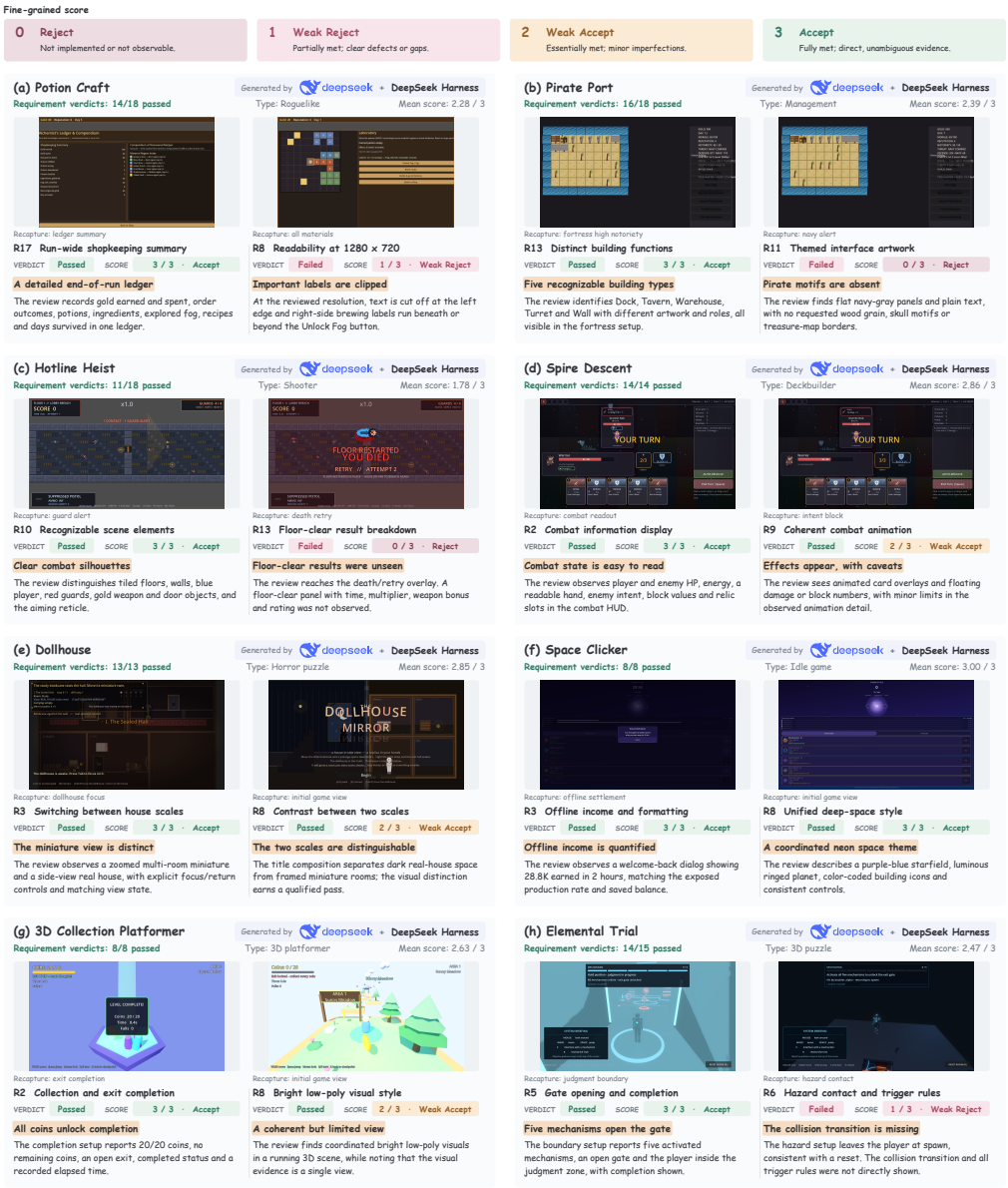


Figure 9: **DeepSeek with the DeepSeek Harness, part 2.** Eight single-round tasks: Potion Craft, Pirate Port, Hotline Heist, Spire Descent, Dollhouse, Space Clicker, 3D Collection Platformer and Elemental Trial. Each task reports its requirement verdicts and the mean rubric score recorded during review, with frames that illustrate the state behind the verdict. The frames are recaptures of the archived build.



Figure 10: **GLM with the ZCode harness, part 1.** Eight single-round tasks: Flappy Bird, Gold Miner, Minesweeper, 3D Snake, Dou Dizhu, Spire Descent, Mini Golf and 3D Collection Platformer. Each task reports its requirement verdicts and the mean rubric score recorded during review, with frames that illustrate the state behind the verdict. The frames are recaptures of the archived build.

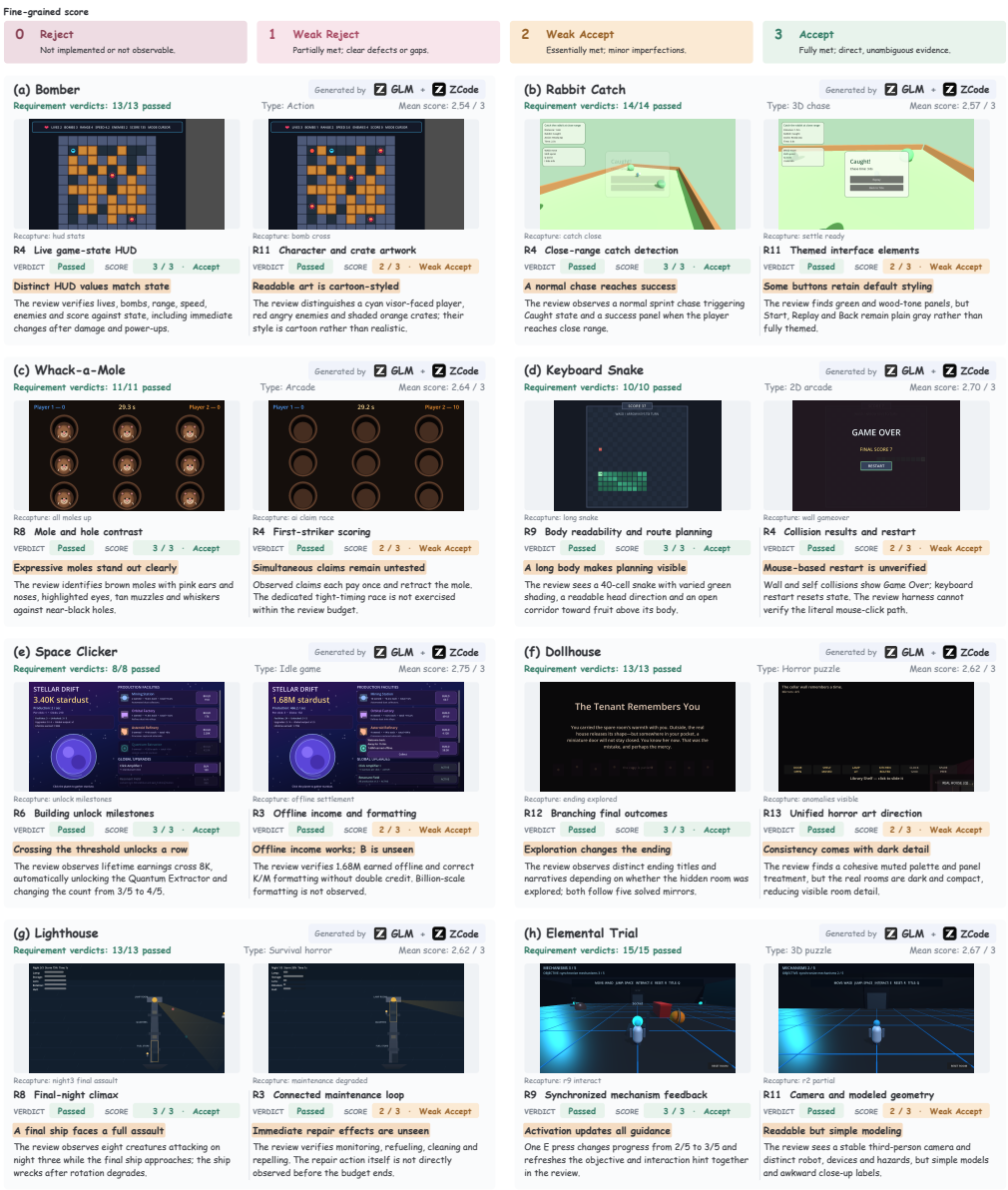


Figure 11: **GLM with the ZCode harness, part 2.** Eight single-round tasks: Bomber, Rabbit Catch, Whack-a-Mole, Keyboard Snake, Space Clicker, Dollhouse, Lighthouse and Elemental Trial. Each task reports its requirement verdicts and the mean rubric score recorded during review, with frames that illustrate the state behind the verdict. The frames are recaptures of the archived build.

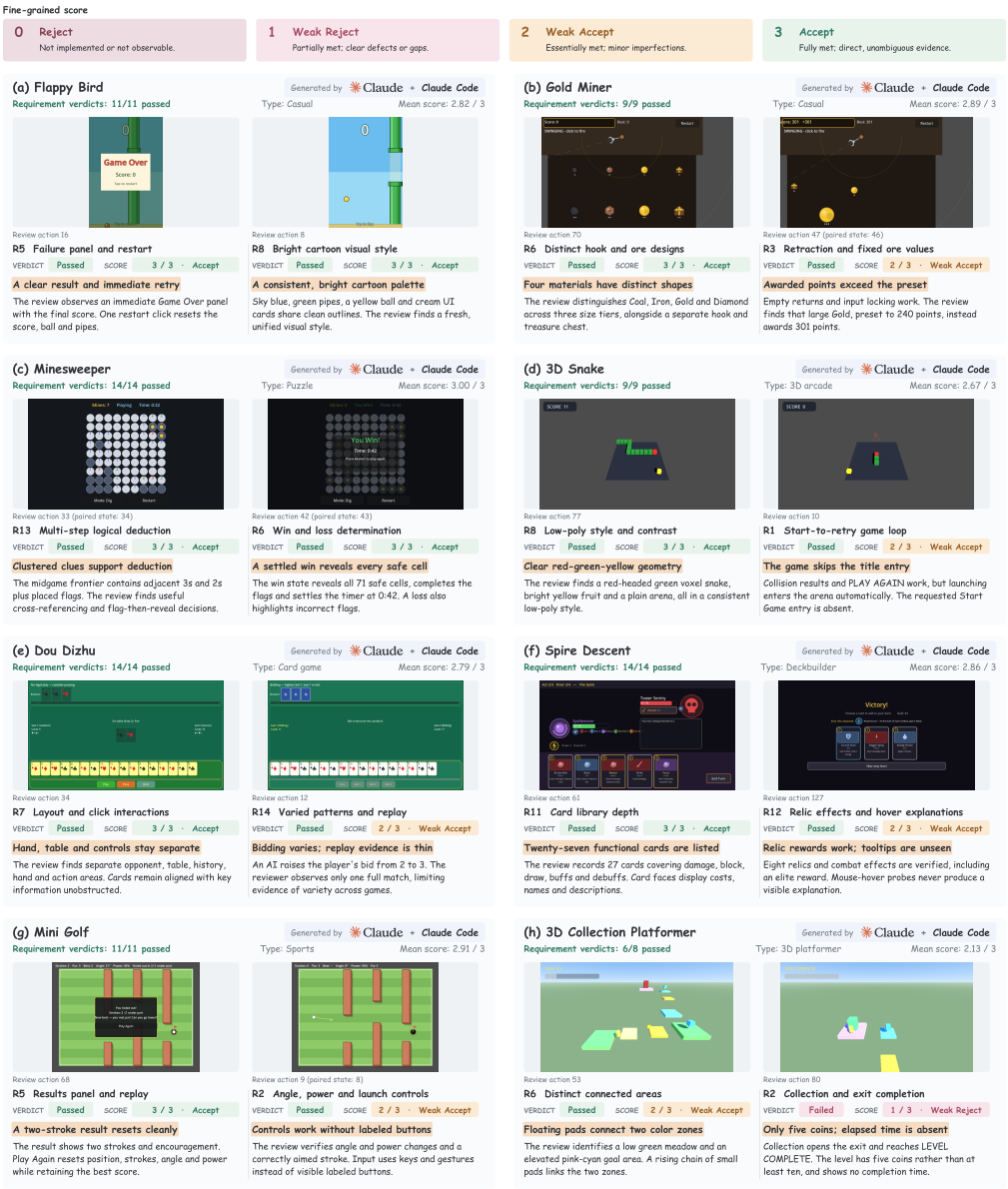


Figure 12: **Claude with Claude Code, part 1.** Eight single-round tasks: Flappy Bird, Gold Miner, Minesweeper, 3D Snake, Dou Dizhu, Spire Descent, Mini Golf and 3D Collection Platformer. Each task reports its requirement verdicts and the mean rubric score recorded during review, with frames that illustrate the state behind the verdict. The frames are the original review screenshots.

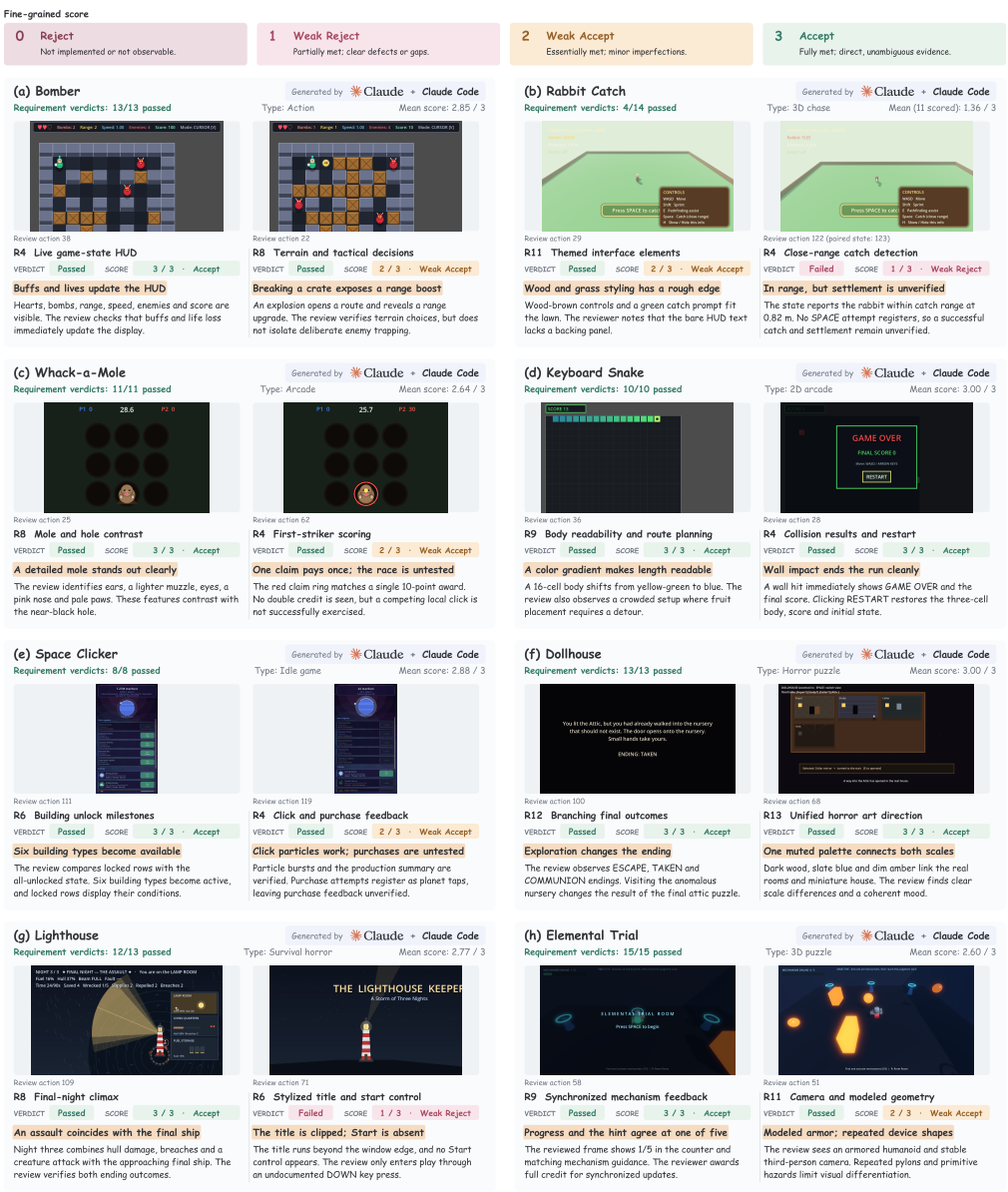


Figure 13: **Claude with Claude Code, part 2.** Eight single-round tasks: Bomber, Rabbit Catch, Whack-a-Mole, Keyboard Snake, Space Clicker, Dollhouse, Lighthouse and Elemental Trial. Each task reports its requirement verdicts and the mean rubric score recorded during review, with frames that illustrate the state behind the verdict. The frames are the original review screenshots.