MMObench: Write the Program That Plays World of ClaudeCraft

Measuring iterative policy authoring in a 20 Hz real-time game

Ryan Foo

`mmobench` (working report)

Leveling runs from July 2026. Dungeon runs and replay audit, November 2026.



Figure 1: World of ClaudeCraft: an open-source, classic-style RPG (nine classes, quests, elite dungeons, trading) simulated deterministically at 20 Hz. An authored policy reads vitals, target geometry, ability readiness, and the five nearest hostiles every step; the quest tracker, minimap, and combat log are omitted. The language model never plays live. It reads logs between runs and revises the program. Leveling tasks are solo; in the party-dungeon track (Section 3.4), one policy controls all five characters.

Abstract

Language models cannot directly play systems that tick faster than model inference. World of ClaudeCraft runs at 20 Hz while an LLM takes seconds to respond, so the only way to play is to write a small JavaScript controller, run it, read what happened, and revise it. We built MMObench to test that loop in a deterministic, open-source world where a 30-minute episode finishes in ≈15 seconds on a laptop. Comparing two ways of using the same model (Gemini 3.5 Flash), the interface dominated everything else: in nine blind prompt-response runs, the model never once beat the hand-coded reference bot (stuck at level 1). Placed in a coding workspace with event journals, greppable game source, and a fast local evaluator, the same model improved in 3 of 3 runs and, in a later campaign, reached held-out level 18 in 39 simulated hours. A third track asks one policy to run a five-character party through an elite dungeon. Its July records (Gemini, 0 of 9 held-out clears) could not be reproduced from committed code and are set aside. On a pinned build in November, two independent Claude Opus 5.5 runs each wrote a policy that clears 5 of 5 held-out seeds, verified by replay; the same policy fails one level lower or without gear, so this is the easiest rung of the task (Section 3.4). Only Gemini completed the agentic leveling track (our Codex arm hit a provider quota before turn 1) and only Claude ran the pinned dungeon track, so nothing here is a cross-vendor ranking. Building the harness also exposed three bugs in our own tooling:

Table 1: Two measured ways of running the authoring loop with the same model (Gemini 3.5 Flash), and the unmeasured system the benchmark grew out of. The workbench rows are separate campaigns with different scores and episode lengths (Appendix B); they are listed together, not pooled.

Interface	What the agent gets	Budget	Measured outcome
Blind edits (classic track)	One summary line per edit (XP, kills, deaths, action histogram); no local test; no source search	20 edits per run	**0 of 6** runs beat p_{ref} held-out with the repaired parser (0 of 9 counting three earlier runs); level 1 (score 5.1 vs. 7.4 for p_{ref})
Coding workbench (agentic track)	`policy.js` in a workspace, event journals, greppable game source, self-serve `eval.mjs`	3 cells × 8 turns per campaign	Campaign 7: **3 of 3** cells improved on training seeds, 2 of 3 beat p_{ref} held-out. Campaign 10 (new score, 72-hour episodes): held-out **level 18** at 39 sim-hours. Campaign 11 (dungeon): 0 of 9 held-out clears, no verified training clear
Live autopilot (origin, not measured)	Rules at 20 Hz, a model setting each character's intent every 8–20 s, a coding agent rewriting the rules between runs	Open-ended, a different model, no held-out seeds	Cleared Gravewyrm Sanctum repeatedly in real time with a coordinated party; no logs of those runs are in the repository

a response parser, a death-penalized score that threw away our fastest policy, and an inverted turn-direction sign. Each looked like model failure until we audited the layer underneath. The environment pin, harness, trajectories, surviving transcripts, and replay tools are being prepared for release.

1 Introduction

Most agent benchmarks test a model *acting*: read a state, pick a move, repeat. That breaks whenever the world does not wait. Trading engines, robot control stacks, and real-time games run loops orders of magnitude faster than LLM latency. There, the agent's job is engineering: write the controller, watch it run, read the logs, and fix it.

World of ClaudeCraft enforces that split by construction. The simulation ticks at 20 Hz, so per-step model inference is impossible. Instead, the model edits a JavaScript function $p : (V, M) \to A$, evaluates it on training seeds, and is scored on held-out seeds it never sees.

Two findings stand out from eleven campaigns of running this loop:

1. **The interface mattered more than the model.** Early runs gave models one compressed summary line per edit and looked like a standard leaderboard: Codex broke through to level 3, while Gemini 3.5 Flash and flagship Gemini 3.1 Pro both failed to beat the hand-coded reference in every run. Moving Gemini 3.5 Flash into a coding workspace, with `policy.js` on disk, per-episode event journals, greppable game source, and a 15-second local evaluator, flipped 0/9 improving runs into 3/3 and set every record in the project; a later campaign with a new score and longer episodes reached held-out level 18 (Table 1).

2. **Three times, the harness manufactured a false verdict.** A broken JSON/fence parser made Gemini look unable to write valid JavaScript; a death-penalized score ($\text{xp}/(1 + \text{deaths})$) rejected the fastest-leveling policy any model had written; and an inverted turn-direction sign made every policy across ten campaigns steer against its own compass (Section 4).

Table 2: The three benchmark tracks. Same deterministic world and held-out scoring; only the authoring interface and task target change.

	classic	**agentic**	**party-dungeon**
model runs as	one completion per edit	coding agent CLI (auto-approved tools)	coding agent CLI
feedback per iteration	one summary line + action histogram + exploration note	per-episode event journals (kills, deaths, level-ups, 2-min windows) + full logs	same, plus per-member combat events
game source	none or fixed 4k-char excerpt	greppable in workspace	greppable in workspace
local test	none between scored edits	self-serve `eval.mjs` (≈15 s at 30 sim-min)	self-serve `eval.mjs`
target	solo level-1 hunter	solo level-1 hunter	one policy for five level-20 characters (warrior, priest, mage, rogue, hunter)
objective	lifetime XP (originally xp/(1 + deaths))	lifetime XP (originally xp/(1 + deaths))	bosses killed, then clear speed; dead stay dead, full wipe ends episode
budget	10–20 edits	8 turns, 15–20 min wall each	8 turns, 20 min wall each

2 The environment and harness

World of ClaudeCraft (Figure 1) is an open-source, server-authoritative RPG with nine classes, levels 1–20, quests, gear, and five-player elite dungeons. One deterministic TypeScript core drives the browser client, the live server, and a headless runner that steps the world over NDJSON (one environment step per five 20 Hz ticks). On a single laptop, a 30 sim-minute episode takes ≈15 seconds of wall clock and an 8 sim-hour episode takes ≈4.3 minutes.

Three properties make the world clean to test against: **determinism** (every edit's effect is reproducible on the exact same seed, unless the policy calls unseeded `Math.random`), **readable source** (the exact simulation code can be granted or withheld), and **freshness** (original maps, formulas, and content not memorized from wikis or walkthroughs).

Policy contract. Across all tracks, a policy is the body of a JavaScript function $p : (V, M) \to A$ over a fixed observation V (vitals, target geometry, five nearest hostiles, interactables, ability readiness, plus a party block in the dungeon track), persistent memory M, and 44 discrete actions. Exceptions or invalid actions count as no-ops. Two fixed geometry quirks are documented in the task card: turning also moves the character forward, and a target's bearing oscillates when directly behind the character, causing naive facing loops to spin in place.

Tracks and floors. Models iterate on three training seeds and are scored on unseen held-out seeds (mean and 95% bootstrap CI over repeats). Two baselines anchor leveling: uniform-random actions and p_{ref}, a hand-written reference bot that also serves as the starting policy. Notably, p_{ref} scores *below* random on held-out seeds (7.4 vs. 23.0 under the early score), showing how easily even hand-coded rules overfit their design seeds.

What an agentic turn looks like. The coding agent wakes in a directory containing `policy.js`, `TASK.md`, the last episode's event journal, the game's source tree, and `eval.mjs`. It reads logs, greps source, edits `policy.js`, and runs local tests before its 15–20 minute turn ends; the harness then scores `policy.js` on the same training and held-out seeds used in the classic track. Across campaigns, agents ran `eval.mjs` 16–70 times and searched the source 10–30 times per cell, pulling out ability buffs and interactable type encodings that appear nowhere in `TASK.md`. In a two-turn pilot with `eval.mjs` removed, the agent shipped a stuck-loop bug both times (`interact` on 100% of steps in turn 1, a 99% `turn_left` spin in turn 2); with `eval.mjs` present, neither shipped.

Protocol changes. Over eleven campaigns (July 2–13, 2026), we fixed three harness bugs, replaced the

Table 3: Classic track with repaired parser (campaigns 5–6: hunter, 20 edits, 30-minute episodes, 3 repeats; held-out mean [95% CI] under xp/(1 + deaths)). One campaign-6 Codex cell was stopped mid-run by the operator and is excluded. Early-protocol floors for scale: random 23.0 [20.3, 25.6]; p_{ref} 7.4 [3.6, 11.1].

model	n	held-out score	level	kills	deaths	compile fails
codex (GPT-5.5, campaign 5)	3	22.6 [3.6, 54.8]	1.3 (best cell L3)	7.6	7.5	0/60
codex (gpt-5.6-sol, campaign 6)	2	12.9 (0, 25.7)	1.0 (best cell L2)	1.8	6.1	0/40
gemini-3.1-pro (campaign 6)	3	5.5 [1.9, 11.1]	1.0	1.1	7.6	0/60
gemini-3.5-flash (campaign 5g)	3	5.1 [1.9, 9.7]	1.0	0.9	7.2	4/60

death-penalized score with lifetime XP, and extended episodes from 21 minutes to 72 hours as bots outran the clock. We never pool raw scores across a metric or decoder change; cross-campaign comparisons use exact deterministic replays on shared held-out seeds (Figure 2), and every campaign's exact condition is logged in Appendix B.

3 Results

3.1 Classic track: compiling code is easy, iterating blindly is not

With one summary line per edit, neither Gemini model beat p_{ref} in a single run: 0 improving runs out of 6 post-repair runs (0 of 9 including pre-repair campaign 5), despite compiling cleanly (4/60 failures for Flash, 0/60 for 3.1 Pro). The failure was concrete: the feedback line flagged low mob visibility and suggested searching when `view.mobs` was empty. Codex acted on that hint (its best cell reached held-out level 3 at 10.2 minutes with 0.7 mobs-visible vs. 0.2 baseline); Gemini ignored it (mobs-visible stayed at 0.1–0.2). Upgrading Flash to flagship 3.1 Pro changed nothing (5.1 vs. 5.5).

Codex was stronger under the blind loop but high-variance: roughly half of all classic LLM runs produced no improving edit at all, and Codex's held-out scores under identical settings ranged from 3.6 to 203.2 in earlier 21-minute runs (Appendix C).

3.2 The agentic reversal: 0/9 to 3/3

In campaign 7, we moved the weakest classic-track model, Gemini 3.5 Flash, into the agentic workspace. It improved on p_{ref} in **all three cells** on training seeds and beat the reference on held-out seeds in two of three (scores 47.6, 248.5, and 4.2, reaching level 3).

The best policy from campaign 7 was one the harness actually threw away. On turn 6, one cell discovered that dying is cheap and reckless grinding wins: it earned 1,083 XP (level 4) on training seeds with 54 deaths. Under xp/(1 + deaths), that scored 18.5 and was discarded as a regression. Replayed on held-out seeds, that discarded policy hit level 4 on 3/3 seeds (level 2 in 2–4 min, level 3 in ≈11 min, level 4 in ≈20 min) and reached the benchmark's first **level 5** at 28.1 sim-minutes (86–99 kills/episode). In 45-minute replays on shared held-out seeds, it hit level 5 on all three seeds (28–32 min) and level 6 on one (43.8 min), while p_{ref} and the best classic Codex policy stalled at levels 1–2.

3.3 Long horizons: fast smoke tests, robustness, and level 18

Verification latency (campaign 9). Stretching episodes to 8 sim-hours (≈4.3 minutes of wall clock) broke the agent's debug loop: waiting 4.5 minutes per local test caused turns 1–2 to time out with 0 XP in all three cells. We patched `eval.mjs` mid-run to accept a minutes override so the agent could smoke-test at 30 minutes and confirm keepers at 8 hours; the next turn jumped to level 6 (1,952 XP). On their own held-out seed families, the three cells reached level 8 (275 min, 859 kills), level 9 (359 min), and level 7 (105 min, 9 deaths). Replayed on a common seed family (Figure 2), however, campaign 7's policy was far more robust than campaign 9's champion, which overfit its seed family.

Level 18 (campaign 10). Switching the score to lifetime XP and extending episodes to 72 nominal hours (with a 120-minute XP-stall cutoff) let aggressive grinding policies compound from turn 1. The best

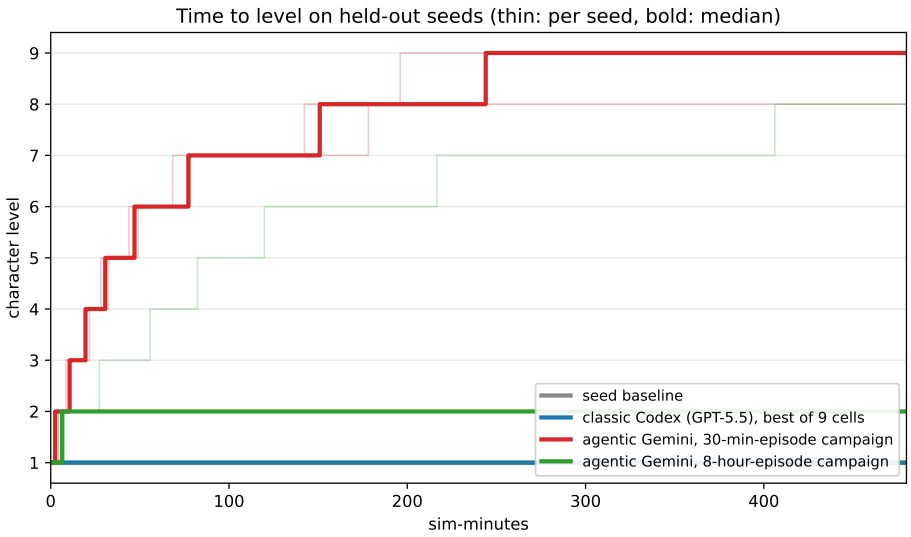


Figure 2: Eight-hour deterministic replays on a shared held-out seed family. The campaign-7 agentic policy (red) climbs to level 9 on two seeds and level 8 on the third; campaign 9's champion (green) hits level 8 on one seed and stalls at levels 1–2 on the others; classic Codex and the baseline stay at levels 1–2.

cell reached **held-out level 18 at 39 sim-hours** (level 5 at 47 min, level 10 at 7.7 hours, level 15 at 23 hours); the other two cells reached levels 11–12, at 280–820 deaths per episode. All of this ran on one laptop via a consumer subscription or a metered API key (campaigns 4–6 combined cost US$2–3 in Gemini API spend). Because the metric change, horizon extension, and bearing-sign fix overlapped here, we report level 18 as an observed ceiling, not an ablation of any single change.

3.4 Party-dungeon track: a verified held-out clear at the easiest setting

The benchmark grew out of a live autopilot that repeatedly cleared the Gravewyrm Sanctum with a coordinated party in real time: rules chose every action at 20 Hz, a model set each character's intent every 8–20 seconds, and a coding agent rewrote the rules between runs. Campaign 11 asked for the same thing under measurement: write *one* policy executed simultaneously by five level-20 best-in-slot characters (warrior, priest, mage, rogue, hunter), deriving roles solely from `view.self.class` and `view.party` (45-minute episodes, fixed bearing decode). We also closed an early corpse-zerging exploit that had let a party clear the dungeon via 49 auto-resurrections; dead characters now stay dead and a full wipe ends the run.

November, pinned build. We rebuilt the party environment from committed code, stored its hash with every result (Appendix A), and ran the track with Claude Opus 5.5 as the coding agent (Claude Code on a consumer subscription, 8 turns of 20 minutes, full source access). We ran two cells with disjoint training seeds in separate workspaces. Table 4 gives the result.

	Cell A (train 1–3)	**Cell B** (train 2001–2003)
Baseline, training seeds cleared	0/3	0/3
After turn 1, training seeds cleared	3/3	3/3
Final policy, held-out seeds 101–105 cleared	**5/5**	**5/5**
Mean held-out clear time (sim-minutes)	4.7	6.1
Mean party deaths per held-out episode	0.6	0.6
Wall clock for the cell (minutes)	54	100

Table 4: Party-dungeon track on the pinned build, Claude Opus 5.5. Both final policies replay to the recorded held-out scores seed for seed.

The two final policies share about 2% of their text, and none of cell B's 407 tool calls referenced cell A's directory, so they are separate solutions. Both cells cleared the training seeds after the first turn; the remaining turns bought speed, not the clear. Figure 3 shows cell A's policy driving the party in the game's own 3D client on held-out seed 101, where it clears in 4.6 minutes with no deaths, matching the headless replay.



Figure 3: The final boss on held-out seed 101, in the game client. One written policy chooses every action for all five characters. The overlay (party panel, caption, boss bar) is drawn by our replay driver from simulation state.

What the policy does. The task prompt includes a short block of general dungeon advice (pull one group at a time, tank first, healer keeps the party alive, assist the tank's target). The July Gemini cells had the same block. Cell A's 299-line policy follows the advice where it holds and departs from it where its own evaluations disagreed: the tank drags each pull back to the party instead of fighting in place; rest before a pull is tiered by what is next (full health before a boss or a pack of three, less before a single elite); it never eats, having measured that food heals more slowly than passive regeneration; a caster whose spell fails for line of sight walks to the tank and retries; and if the tank dies the next living member takes over pulling. Both cells also found a flaw in our interface: the action set has no way to target an ally, so the priest can heal only itself, and the prompt's instruction to heal the lowest member cannot be carried out.

How much is the party's strength? Most of it. The track fixes a maximum-level party in best-in-slot gear. Replaying cell A's final policy on the held-out seeds with a weaker party gives: level 20 with gear, 5/5 clears; level 20 with no gear, 0/5; level 19 with gear, 1/5; level 18 with gear, 0/5. The gear alone does not clear (the baseline has it and wipes at about two minutes), but the written policy has almost no margin, and it was developed against exactly these stats. We read the result as the easiest rung of the task being solved, and level 19 or no gear as the next rung, which nothing passes yet.

Caveats on the November cells. One model. The agent was told to stay inside its workspace and did not: 27 of cell B's tool calls read the harness's runner code or the environment checkout. None touched the scorer's outputs or the results directory, and the held-out seeds are fixed and public in the harness, so the claim rests on replay of the final policy, not on isolation.

What was recorded in July, and why it is set aside. The naive baseline wipes at 2.1 sim-minutes. Over 8 turns, agentic Gemini 3.5 Flash wrote tank-tracking, party-wide combat state, and mana-gated pull pacing. Two of three cells never improved on the baseline. On turn 8 the third cell recorded a clear on one training seed (score 3,478; the event log shows the boss killed at 6.9 sim-minutes with one party death). On held-out seeds the record is **0 of 9** (cells $\times$ seeds).

What a replay audit found (November 2026). The built party environment from July was not preserved, so we rebuilt it from the two committed dungeon commits and replayed the stored policies. The baseline reproduces exactly on three training seeds (score 10.0, 0.7 wipes, 3.3 deaths). None of the six model-written policies we replayed reproduces its recorded score. The policy that recorded the clear scores 11.0

Table 5: Harness failures that impersonated model failure.

Failure	False verdict	Reality after fix
Response parser (JSON/fence extraction, campaigns 4–5)	"Gemini cannot write valid JS" (near-zero compile rate)	Replayed on 37 failing outputs, parser recovered 35/37; live compile fails dropped to 4/60 (Flash) and 0/60 (3.1 Pro). Iteration still failed cleanly.
Death-penalized score (xp/(1 + deaths), retired campaign 10)	Campaign 7 turn-6 policy "regressed" (18.5) and was discarded	Fastest-leveling policy in the repo: held-out level 4 on 3/3 seeds, first level 5 at 28.1 min. Replaced by lifetime XP.
Inverted turn sign (observation decoder, campaigns 1–10, fixed `d1441aa`)	Models "struggle with facing"; depressed absolute scores	At bearing $-14°$, `turn_right` closed the angle ($\approx 45°$/step). Every policy in campaigns 1–10 steered against its compass. Uniform across models, so within-stratum rankings held.

with no clear on the final commit (`de006a66e`, the dead stay dead). On the commit before it (`ca9f3408c`, auto-resurrect still on) the same policy clears all three training seeds with 27–43 deaths each, which is the exploit, not the recorded result. Replays are deterministic, so the July runs used a build that differed from both commits in ways only the richer policies exercise. We cannot say how.

We therefore treat campaign 11's numbers as unverified and do not compare them with the November cells: the builds differ, so the gap between Gemini's July record and Claude's November result is not a model comparison.

4 Three times the harness was wrong

Three separate harness defects looked like model incapability until we audited the code underneath (Table 5); four smaller incidents (400-token truncation, CLI timeouts, empty-string quota errors, and parameter-shadowing in a helper ablation) are logged in Appendix D.

The intake and decode bugs survived until an all-fail result forced an audit. The score bug was subtler: nothing crashed, the metric simply penalized the exact strategy that won, and only an agent optimizing the real goal exposed it. The inverted compass survived ten campaigns because it hurt every model equally. Winning policies quietly adapted by abandoning targets behind them and retargeting while moving. Our standing audit rule: on any all-fail run, check the parser and provider path first; on any metric, check what a goal-directed agent would do that the metric punishes; and on any decoded observation, test the documented sign against the simulator empirically.

5 Related work

Voyager [1] writes Minecraft skill libraries, but Mineflayer skills execute while the model waits and Minecraft's source cannot be toggled as an experimental variable. Code as Policies [18] and Eureka [19] operate at the same authoring layer in robotics because LLMs cannot sit inside a servo loop. Programmatic-policy RL [3] and code world models [2] share the shift from acting to building programs; SWE-bench [8] is the template for our read-edit-verify workspace, with embodied game performance replacing unit tests. Unlike AlphaStar [9] and OpenAI Five [10] (up to 200 and ≈45,000 years of simulated experience), MMObench prices marginal per-task authoring with a pretrained model producing readable code on one laptop. Other turn-based or human-speed benchmarks and agents (Vending-Bench [6], AgentBench [7], Cradle [4], SIMA [5], Crafter [11], MineRL [12], MineDojo [13], NLE [14], Melting Pot [15]) do not force code-as-policy by loop rate. Reflexion [16] and Self-Refine [17] supply the act, reflect, revise loop that determinism lets us attribute edit by edit.

6 Limitations and what comes next

The main gap in this report is cross-vendor agentic data: our Codex agentic arm hit a provider quota before turn 1, no Claude arm has run on the leveling tracks, and only Claude has run the pinned dungeon track. The dungeon clears hold only for the strongest party the game allows (Section 3.4). The interface jump also bundled workspace, event journals, greppable source, and local verification together, and turn budgets cap wall clock rather than edit count (which penalizes slower models). Sample sizes are small (2–3 cells per condition) on a single solo class (hunter) in one game. The dungeon track's July numbers could not be reproduced from committed code (Section 3.4); the leveling tracks are not affected, since their built environment was preserved byte for byte and stored policies replay on it. The missing runs (cross-vendor agentic arms, a source-access and evidence ablation, a parameter-tuning vs. code-structure check, and the dungeon track with other models and weaker parties) are listed in Appendix E. All runs target local, experimenter-controlled servers; nothing here touches a live operator's game.

7 Conclusion

When a world runs at 20 Hz, an LLM cannot be the player. It has to be the engineer. In World of ClaudeCraft, giving the weakest blind-loop model a real engineering workbench turned 0/9 improving runs into 3/3, and a later campaign reached held-out level 18. Along the way, our own parser, metric, and observation decoder each produced false verdicts that looked identical to model failure, and a replay audit showed that one headline number, a July dungeon clear, could not be reproduced. On a pinned build, one written policy then ran a five-character party through the Gravewyrm Sanctum on five of five held-out seeds, in two independent runs, at the strongest party setting. The environment pin, harness, policies, trajectories, surviving transcripts, and replay tools are being prepared for release so that anyone can rerun these results and take a shot at the open challenge: the same clear one level lower, or without gear.

References

[1] G. Wang et al. Voyager: An Open-Ended Embodied Agent with Large Language Models. *TMLR*, 2024 (arXiv:2305.16291, 2023).

[2] W. Lehrach et al. Code World Models for General Game Playing. arXiv:2510.04542, 2025.

[3] A. Verma et al. Programmatically Interpretable Reinforcement Learning. *ICML*, 2018.

[4] W. Tan et al. Cradle: Empowering Foundation Agents Towards General Computer Control. arXiv:2403.03186, 2024.

[5] SIMA Team. Scalable Instructable Multiworld Agent. DeepMind technical report, 2024.

[6] A. Backlund and L. Petersson. Vending-Bench: A Benchmark for Long-Term Coherence of Autonomous Agents. arXiv:2502.15840, 2025.

[7] X. Liu et al. AgentBench: Evaluating LLMs as Agents. *ICLR*, 2024.

[8] C. Jimenez et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *ICLR*, 2024.

[9] O. Vinyals et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 2019.

[10] OpenAI. Dota 2 with Large Scale Deep Reinforcement Learning. arXiv:1912.06680, 2019.

[11] D. Hafner. Benchmarking the Spectrum of Agent Capabilities. *ICLR*, 2022.

[12] W. Guss et al. MineRL: A Large-Scale Dataset of Minecraft Demonstrations. *IJCAI*, 2019.

[13] L. Fan et al. MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge. *NeurIPS*, 2022.

[14] H. Küttler et al. The NetHack Learning Environment. *NeurIPS*, 2020.

[15] J. Leibo et al. Scalable Evaluation of Multi-Agent Reinforcement Learning with Melting Pot. *ICML*, 2021.

[16] N. Shinn et al. Reflexion: Language Agents with Verbal Reinforcement Learning. *NeurIPS*, 2023.

[17] A. Madaan et al. Self-Refine: Iterative Refinement with Self-Feedback. *NeurIPS*, 2023.

[18] J. Liang et al. Code as Policies: Language Model Programs for Embodied Control. *ICRA*, 2023.

[19] Y. J. Ma et al. Eureka: Human-Level Reward Design via Coding Large Language Models. *ICLR*, 2024.

A Task card, prompts, and provenance

The complete task card (decoded observation schema, 44-action set, movement realities, scoring protocol) and authoring prompts are fixed per track and reproduced verbatim in the repository. All leveling-track results (strata P0–P4) used env bundle `sha256 58aa0ab5...` on Node v24.13.0. That built file is preserved; no upstream commit rebuilds to the same hash, so the binary and its hash are the pin. The dungeon track (P5) drove a party-enabled build from the game's `dungeon-env` branch (upstream `ac80a992e` plus `ca9f3408c` and `de006a66e`). That built file was not preserved, and the dungeon records carry the leveling bundle's hash, not its own. The pin for future dungeon runs is a rebuild of `de006a66e` with esbuild 0.28.1. Every results record carries a bundle hash and harness commit. Artifacts live under `bench/results/campaign{4,5,5g,6}/`, `bench/results/agentic/`, and `bench/results/dungeon/`; agentic per-turn transcripts survive for campaigns 10–11 (campaigns 7 and 9 reused the same cell workspaces and were overwritten by campaign 10, leaving campaign-level logs). Campaigns 1–2 and valid campaign-3 cells are under `research/results/trajectories/`; the dated ledger is `research/EXPERIMENT-LOG.md`.

B Per-campaign condition matrix and protocol strata

Table 6 lists every campaign with its stratum (P0–P5, grouping comparable score, horizon, and decode conditions), track, models, budgets, horizon, score, and deviations.

C Early classic campaigns (stratum P0) and ablations

Campaign 1 (10-edit budget, n=2, held-out means with 95% CIs): codex/GPT-5.5 103.4 [3.6, 203.2] with fixed 4k-char source excerpts and 99.3 [79.3, 119.3] without; random floor 23.0 [20.3, 25.6]; gemini-2.5-pro 11.5 (source on) and 7.4 (off); p_{ref} and qwen2.5-coder-3B both 7.4 [3.6, 11.1]. Campaign 2 (motor-primitives ablation): given a library of proven navigation helpers (adopted in every helpers-on edit), gemini-2.5-flash still never beat p_{ref} (best held-out mean 6.7 vs. 7.4), and codex's apparent helpers-on dip (11.5 vs. 33.2 off) is unreadable at n=2 given within-condition train spread of 433 vs. 17. Overfitting was the norm: the largest measured train-minus-held-out gap was 373.6 (train 433.3, held-out 59.7). In campaign 1, gemini-2.5-pro's policies compiled cleanly yet spent 86% of steps oscillating `turn_left`/`turn_right` on the bearing trap, scoring 0 XP across every edit even when shown the oscillation trace.

D Incident ledger

(i) A 400-token default cap truncated policies mid-statement; a 3B model's compile rate rose from 8% to 67% after wrapper-robust parsing and truncation retry. (ii) A 120 s CLI timeout silently consumed up to two of ten iterations per cell for Gemini CLI models; raising it to 240 s preceded that family's first classic-track gain (held-out 58, five improving edits of twelve, campaign 3-partial). (iii) Provider quota exhaustion returned empty strings that were scored as 24 compile failures until audit; empty outputs are now rejected as provider errors (which is why campaign 8 aborted cleanly with zero turns). (iv) The helpers ablation induced a parameter-shadowing error (models redeclaring `const lib`), now recovered at intake.

Table 6: Every campaign, condition, and deviation. Score "XP/d" is xp/(1 + deaths); "LXP" is lifetime XP. Bearing: I = inverted decode, M = mixed (fix landed mid-campaign), F = fixed. Campaigns 4–11 ran July 9–13, 2026.

str.	camp.	track	model(s)	cells	budget	episode	score	bearing / deviations
P0	1	classic	codex GPT-5.5; gemini-2.5-pro; qwen-3B	20	10 edits	21 min	XP/d	I; source on/off ablation
P0	2	classic	codex GPT-5.5; gemini-2.5-flash	8	10 edits	21 min	XP/d	I; helpers on/off ablation
P0	3	classic	(partial)	—	12 edits	21 min	XP/d	I; codex quota incident; partial versioned
P1	4	classic	codex GPT-5.5; gemini-3.5-flash	6	20 edits	21 min	XP/d	I; gemini cells parser-confounded
P1	5	classic	codex GPT-5.5; gemini-3.5-flash	6	20 edits	30 min	XP/d	I; flash cells pre-parser-fix
P1	5g	classic	gemini-3.5-flash	3	20 edits	30 min	XP/d	I; parser repaired (json ON + extraction)
P1	6	classic	codex gpt-5.6-sol; gemini-3.1-pro	5	20 edits	30 min	XP/d	I; 3rd codex cell operator-stopped
P2	7	agentic	gemini-3.5-flash	3	8 turns × 15 min	30 min	XP/d	I; source grep on
—	8	agentic	codex gpt-5.6-sol	3	8 turns	30 min	XP/d	aborted at launch (quota); zero turns
P3	9	agentic	gemini-3.5-flash	3	8 turns × 20 min	8 h	XP/d	I; full source; mid-run eval.mjs minutes-override hot-patch
P4	10	agentic	gemini-3.5-flash	3	8 turns	72 h, 120-min stall	LXP	M; score+horizon changed together
P5	11	dungeon	gemini-3.5-flash	3	8 turns × 20 min	45 min	bosses, speed	F; auto-res exploit closed; not reproducible
P6	12	dungeon	claude-opus-5-5	2	8 turns × 20 min	45 min	bosses, speed	pinned build; replay-verified

E Planned next runs

(1) Cross-vendor agentic arms (Codex and Claude under the campaign-7 protocol). (2) Evidence-abstraction ablation (raw step dumps vs. decoded view vs. event journals vs. summaries vs. the live autopilot's telemetry bundle). (3) Structure vs. dials (coordinate descent on numeric literals to separate new logic from tuned constants). (4) One-shot vs. iterated and budget scaling on the classic track. (5) Class generality beyond hunter. (6) The dungeon track on the pinned build with other models, with a workspace the agent cannot read out of, and at the weaker party settings (level 19, no gear) where no policy clears yet.