RTSGAMEBENCH: EVALUATING VLMS FOR STRATEGIC REASONING IN GAME ENVIRONMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language Models (VLMs) often struggle with *strategic reasoning* in a multi-agent setting, *i.e.*, deciding under uncertainty while coordinating with and adapting to other agents. Real-time strategy (RTS) games can be a natural testbed for this capability, requiring long-horizon planning, coordination with allies, and adaptation to opponents under partial observability. But existing RTS benchmarks cover limited agent configurations, lack controlled evaluation across distinct strategic demands, and remain confined to fixed scenarios. To address these gaps, we propose RTSGAMEBENCH, built on *Beyond All Reason*, an RTS game whose larger scale broadens the strategic space beyond prior testbeds. In RTSGAMEBENCH, we offer holistic evaluation across diverse full-game matchups, diagnostic mini-games each emphasizing a focal strategic demand, and extensible coverage via a self-evolving framework that turns free-form queries into new mini-games, improving with accumulated experience. We further propose RTSGAMEAGENT, an agent harness that combines scalable group-level trajectory planning with short- and long-term memory and outperforms existing agents. We find that state-of-the-art VLMs struggle where humans succeed, particularly when they must coordinate with allies or contend with multiple opponents.

1 INTRODUCTION

Vision-Language Models (VLMs) have achieved remarkable success across diverse tasks (Alayrac et al., 2022; Liu et al., 2023), yet deploying them in complex, evolving environments requiring long-horizon decisions in the presence of other agents remains challenging (Paglieri et al., 2025). A key capability in such environments is *strategic reasoning*, which involves deciding under uncertainty while coordinating with and adapting to other agents, both competitors and collaborators (Gandhi et al., 2023; Zhang et al., 2024). We argue that real-time strategy (RTS) games are a natural testbed as they require long-horizon decision-making under partial observability, including resource allocation, multi-unit coordination, adaptation to opponents, and cooperation with allies, within a measurable and reproducible simulator (Ontañón et al., 2013; Robertson & Watson, 2014).

While StarCraft II (SC2) has been widely used as an RTS testbed for AI research (Vinyals et al., 2019; Ma et al., 2024; Li et al., 2025), we build upon *Beyond All Reason* (BAR) (Beyond All Reason Team, 2024) featuring larger battles with more units and larger maps than SC2 (Tab. 1; see Appendix A.2). This expanded scale naturally broadens the strategic space, requiring longer-horizon planning over many interacting units, coordination across allied groups, and reasoning about enemy formations (Ontañón et al., 2013). Meanwhile, BAR partially automates low-level execution, from target selection and pathfinding to energy conversion (Appendix A.1), reducing low-level overhead while leaving the agent to decide how to allocate units, when and where to engage, and how to coordinate spatial maneuvers. Together, BAR's scale and partial low-level automation make it a suitable platform for evaluating VLMs' strategic reasoning in RTS.

However, the platform alone does not guarantee a rigorous evaluation. Strategic reasoning in RTS is inherently multi-faceted (Buro, 2003), spanning resource management, spatial and temporal reasoning, opponent modeling, collaboration, adversarial planning, and decision making under uncertainty. Moreover, these strategic demands vary with the number, roles, and relationships of agents. But existing benchmarks cover this space only partially. They consider limited agent configurations, lack controlled evaluations across distinct strategic demands, and remain confined to fixed scenarios (Ma

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.

Table 1: **Quantitative comparison of StarCraft II (SC2) *vs*. BAR.** *Unit Variety*: unique units and buildings across all factions. *Supply Cap*: per-player unit limit. *Unit Capacity*: total unit limit across all players. *SC2 uses a weighted population system, so actual unit counts are lower than this.

	Unit Variety	Supply Cap	Unit Capacity	Map Size	Player Limit
StarCraft II (SC2)	96	200*	1,600*	1×	8
Beyond All Reason (BAR)	**554**	**2,000**	**32,000**	**64**×	**100**

et al., 2024; Ahn et al., 2025; Li et al., 2025) (see Sec.2 and Appendix B for detailed comparisons). To this end, we argue that a rigorous RTS benchmark should be: (i) *holistic*, measuring integrated strategic performance in complete games across diverse matchup structures; (ii) *diagnostic*, using controlled scenarios to probe different strategic demands; and (iii) *extensible*, allowing diagnostic coverage to expand beyond a fixed scenario set via automated, on-demand generation.

To address the desiderata, we propose **RTSGAMEBENCH**, a benchmark with three components (Fig. 1): (i) *Full Games*, evaluating the agent against and alongside built-in AIs across diverse match-ups; (ii) *Diagnostic Mini-Games*, controlled scenarios organized around focal strategic demands, grounded in a taxonomy of RTS AI challenges (Buro, 2003); and (iii) a *Game Generation Framework*, converting free-form user queries into diagnostic mini-games. As this framework expands diagnostic coverage via repeated generation, treating each generation cycle independently would discard the validated artifacts and experience accumulated in earlier cycles, leading to redundant regeneration and recurring failures. We therefore introduce *experience-driven evolution*, which makes the framework self-evolving by reusing these artifacts and experiences to progressively improve generation efficiency and quality.

In addition, despite BAR's partial low-level automation, two challenges remain for deploying VLMs at this scale. Growing combat forces make per-unit control increasingly costly, a challenge compounded by sparse VLM calls that limit how often unit behavior can be revised. Meanwhile, long-horizon play requires integrating recent events with longer-term context. To address these, we propose **RTSGAMEAGENT**, a baseline VLM agent harness that organizes combat units into groups and plans trajectories of group commands to sustain execution between VLM calls while maintaining temporal context via short- and long-term memory. RTSGAMEAGENT outperforms competing agents under matched settings. With it, we find state-of-the-art (SoTA) VLMs struggle where humans succeed, especially when coordinating with allies or contending with multiple opponents, and mini-games suggest that their relative strengths vary across strategic demands. Finally, experience-driven evolution improves generated games while reducing token usage.

2 RELATED WORK

Game-based evaluation of LLM and VLM agents. Games serve as effective testbeds for evaluating the decision-making capabilities of LLM- and VLM-based agents (Paglieri et al., 2025; Hu et al., 2026; Park et al., 2026). Early benchmarks relied on textual observations in single-agent (Hu et al., 2024) or multi-agent strategic settings (Qi et al., 2024), while multimodal benchmarks such as EscapeCraft (Wang et al., 2025) and MCU (Zheng et al., 2025) remain largely single-agent. As full-game outcomes can obscure the sources of success or failure (Lin et al., 2025), recent studies adopt scenario-level evaluation (Tang et al., 2026; Zheng et al., 2026) or separate strategic reasoning from decision-making (Xu et al., 2026), mostly over predefined scenarios. Scenario generation has been explored for single-agent tasks (Zheng et al., 2025) and text-based two-player games (Verma et al., 2025), but without targeting strategic demands specified by users. RTSGAMEBENCH instead evaluates VLM agents in a large-scale multi-agent RTS environment and extends its diagnostic mini-games on demand from free-form user queries.

RTS game benchmarks. Most RTS benchmarks for LLM-based agents are built on SC2: TextStarCraft II (Ma et al., 2024), TEXTSCII-ALL (Ahn et al., 2025), and HIVE (Anne et al., 2025) use textual observations, while AVACraft (Ma et al., 2026), LLM-PySC2 (Li et al., 2025), and Kim et al. (2026) introduce multimodal scenario- or full-game evaluation. Most closely related, Li et al. (2026) introduce a diagnostic long-horizon RTS benchmark for LLM agents in Red Alert. RTSGAMEBENCH differs by evaluating multimodal VLMs across diverse multiplayer match-ups and taxonomy-grounded strategic demands, combining complete games with targeted diagnostics

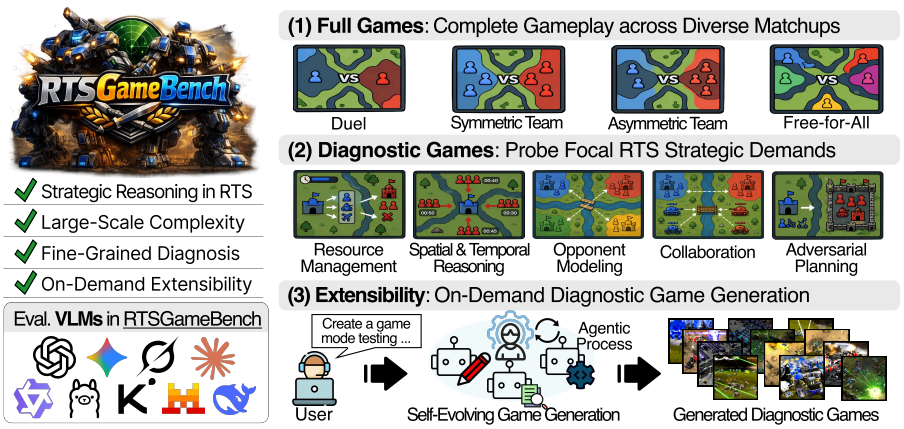


Figure 1: **Overview of RTSGAMEBENCH.** The benchmark comprises (1) *Full Game Evaluation* across diverse matchups, (2) *Diagnostic Mini-Games* designed for different focal strategic demands, and (3) a *Self-Evolving Game Generation Framework* for on-demand diagnostic game generation.

while enabling on-demand diagnostic game generation beyond a fixed scenario set; building on BAR also supports substantially larger-scale evaluation (see Appendix B for a detailed comparison).

LLM-based game and environment generation. LLMs have automated game development with role-specialized multi-agent frameworks (Chen et al., 2023) and generated executable environments for open-ended learning (Faldor et al., 2025) or cross-environment agent learning (Zhang et al., 2025). Our framework similarly builds on role-specialized agents (Hong et al., 2024; Chen et al., 2023), but targets diagnostic evaluation, designing each game around a focal strategic demand in a complex RTS engine and validating it via staged rubric checks and simulation tests. It also evolves across generations. Unlike archives that seed novel tasks (Faldor et al., 2025) or agents that self-improve on the tasks they solve (Shinn et al., 2023; Zhao et al., 2024), it reuses validated designs and rule code to skip redundant stages and refines its rubrics through retrospectives.

3 RTSGAMEBENCH

As discussed in Sec. 1, rigorous evaluation of strategic reasoning in RTS requires a holistic, diagnostic, and extensible benchmark, which existing benchmarks partially meet. To address this, we introduce RTSGAMEBENCH, built on BAR, with three components (Fig. 1): (i) *Full Games* (Sec. 3.1), (ii) *Diagnostic Mini-Games* (Sec. 3.2), and (iii) *Game Generation Framework* (Sec. 3.3).

Interface. All inferences follow an observe–decide–act loop. The agent uses game knowledge $\mathcal{K}$, including the scenario description, available units and buildings, and team configuration. At each interval t, visual observations v_t (global minimap and local camera views) are captured from game state s_t, while a Python wrapper $\mathcal{W}$ extracts textual state $\mathcal{W}(s_t)$ to form multimodal observation o_t. With Fog-of-War (FoW)[1] enabled, the observation o_t is restricted to allied line of sight. Given $\mathcal{K}$ and o_t, the agent's VLM policy π produces action a_t, which updates the environment state:

$$o_t = (v_t, \mathcal{W}(s_t)), \quad a_t = \pi(o_t \mid \mathcal{K}), \quad s_{t+1} \leftarrow \text{Env}(s_t, a_t). \tag{1}$$

The action space comprises *building construction*, *unit production*, and *unit action*. The agent decides which building produces each unit, which unit constructs each building, and where to build or move on a $(0, 0)$–$(100, 100)$ grid, while Env handles low-level execution. The game runs for a fixed interval between decisions and pauses during inference, so evaluation reflects strategic decision quality rather than reaction speed. Full interface and $\mathcal{K}$ details are provided in Appendix C.

3.1 FULL GAME MATCH-UPS

In full-game evaluation, an agent plays complete BAR matches. While existing RTS benchmarks mainly focus on 1v1 settings (Ma et al., 2024; Li et al., 2025), different player configurations impose distinct competitive and cooperative demands (Buro, 2003). We thus evaluate four match-ups

[1] A game mechanic that hides areas outside allied units' line of sight, introducing partial observability.

Table 2: **Evaluation settings in RTSGAMEBENCH.** *Top:* Full-game matchups. *Bottom:* Diagnostic mini-games designed to make different focal strategic demands particularly salient (Buro, 2003). Action types (Sec.3): Build = building construction, Prod. = unit production, Action = unit action.

Full Game Match-ups				
Mode	**Config**	**Strategic Demand**	**Action Type**	**FoW**
Duel	1v1	Individual decision-making	Build + Prod. + Action	On
Symmetric Team	2v2	Allied coordination	Build + Prod. + Action	On
Asymmetric Team	3v4	Coordination under numerical disadvantage	Build + Prod. + Action	On
Free-for-All	1v1v1v1	Multi-polar threat prioritization	Build + Prod. + Action	On
Diagnostic Mini-Games				
Focal Strategic Demand	**Game**	**Task**	**Action Type**	**FoW**
Resource Management (RM)	TCP	Meet production and economy targets	Build + Prod. + Action	On
Spatial&Temporal Reasoning (S&T)	MFD	Defend multiple objectives from staggered attacks	Action	Off
Opponent Modeling (OM)	FBA	Analyze and destroy a fortified enemy base	Build + Prod. + Action	On
Collaboration (COL)	ATS	Form a team with allied forces and fight cooperatively	Prod. + Action	Off
Adversarial Planning (AP)	AWD	Survive enemy waves whose composition is known in advance	Build + Prod. + Action	Off

(Tab. 2, top): *Duel* tests individual capability, *Symmetric Team* adds allied coordination, *Asymmetric Team* requires coordination from the smaller team, and *Free-for-All* requires prioritizing threats among multiple opponents. In all settings, the evaluated agent occupies one slot and all remaining slots are filled by built-in AIs at fixed difficulty levels.

3.2 DIAGNOSTIC MINI-GAMES

Full-game evaluation requires multiple strategic demands simultaneously, making it difficult to attribute performance patterns to any particular demand. We thus introduce controlled diagnostic mini-games (Tab. 2, bottom), each organized around a focal demand from a taxonomy of RTS AI challenges (Buro, 2003), at three difficulty levels. Rather than strictly isolating capabilities, these scenarios make one demand particularly salient while preserving the coupled decision-making inherent to RTS gameplay. Human ratings support this, as each scores highest on its focal demand while still eliciting others (Fig. 2; Appendix E.1). We thus interpret them as targeted probes of behavior under particular demands, rather than isolated measurements of latent capabilities, providing a finer-grained characterization of model behavior than full-game outcomes alone, as follows:



Figure 2: **Strategic demands of each mini-game.** Mean human ratings (1–5).

(1) *Resource Management: Time-Constrained Production (TCP).* The agent must meet economy and production targets within a time limit. Production dependencies (Qualitipedia contributors, 2024) force sequential build decisions, testing how it balances economic growth against unit production.

(2) *Spatial & Temporal Reasoning: Multi-Front Defense (MFD).* With fixed forces and no production, the agent defends multiple buildings against attacks arriving from different directions at staggered times, placing emphasis on maneuvering forces across fronts as threats emerge.

(3) *Opponent Modeling: Fortified Base Assault (FBA).* The agent must destroy a key structure inside a fortified enemy base whose defenses are revealed only via scouting, placing emphasis on inferring the enemy's defensive setup and adapting forces and approach accordingly.

(4) *Collaboration: Allied Team Skirmish (ATS).* With building construction disabled, the agent fights alongside a built-in AI ally in a 2v2 battle, emphasizing coordination with this ally; the agent wins if its team's score, combining damage dealt and forces preserved, exceeds that of the opposing team.

(5) *Adversarial Planning: Anticipatory Wave Defense (AWD).* The agent must protect its commander from successive waves whose unit compositions are known in advance, emphasizing anticipatory counter-planning for future enemy attacks.

Another challenge in the taxonomy, *Decision Making under Uncertainty*, is incorporated selectively via FoW (Tab. 2, FoW column). See Appendix E for mini-game details and difficulty levels.

3.3 GAME GENERATION FRAMEWORK

Each RTS strategic demand can arise in diverse scenarios that a fixed mini-game set cannot fully cover. Expanding this coverage manually is costly, as game creation spans stages such as scenario design, engine-level implementation, and simulation-based validation, each requiring distinct ex-

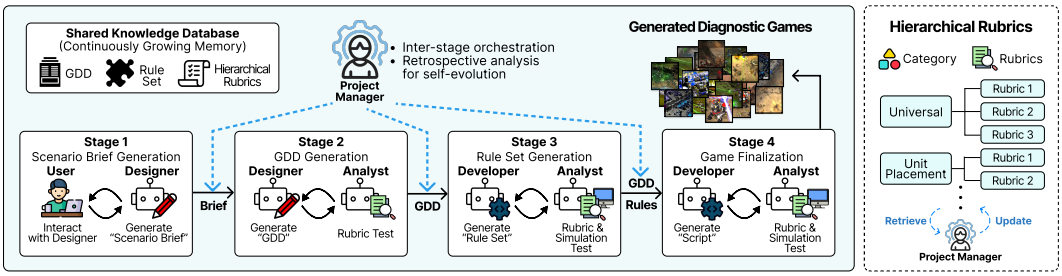


Figure 3: **Self-evolving game generation.** The PM coordinates four stages with rubric retrieval, retries, and rollback, while shared experience enables reuse and rubric updates across generations.

pertise (Hendrikx et al., 2013); this motivates generating diagnostic games automatically and on demand. Moreover, experience from earlier generation cycles can be reused to avoid redundant effort and recurring failures. With both in mind, we propose a *self-evolving* game generation framework that builds mini-games from free-form queries and progressively improves its efficiency and quality through accumulated experience (Fig. 3; Alg. 2 in Appendix H.1).

Generation pipeline. Handling all stages in a single agent would mix heterogeneous contexts (Liu et al., 2024; Shi et al., 2023) and obscure where failures originate. We therefore take a *divide-and-conquer* approach with a *role-specialized multi-agent system* (Hong et al., 2024): a *project manager* (PM) for orchestration, a *designer* for game design, a *developer* for implementation, and an *analyst* for validation. The PM coordinates four stages, rolling back to the stage responsible for each validation failure and aborting once a fixed rollback budget is exhausted (Appendix H.1). Before each validation, the PM retrieves relevant *validation rubrics* from a shared knowledge database. In *Stage 1*, the designer interacts with the user to produce a scenario brief specifying game composition, enemy behavior, and win/loss conditions; if the database contains a matching game design document (GDD) (Fullerton, 2014) and rule set, the pipeline skips to *Stage 4*. In *Stage 2*, the designer expands the brief into a GDD defining the focal demand, game rules, and a checklist for the final game, and the analyst validates it. In *Stage 3*, the PM retrieves stored rules, the developer writes the rest in Lua (Ierusalimschy, 2006) with rule-based simulation tests, and the analyst validates the rule set against the tests and rubrics. In *Stage 4*, the developer builds an executable game script with tests, which the analyst validates against the tests, rubrics, and GDD checklist. The validated script is then scaled to three difficulty levels, using the mini-games in Sec.3.2 as references.

Experience-driven evolution. Rather than treating each generation independently, the framework accumulates and reuses experience in two ways (validated in Sec.5.3). First, validated GDDs and rule sets are stored in the shared knowledge database and reused when applicable, bypassing redundant stages. Second, after each successful generation, the PM performs a *retrospective analysis* of its generation history, including failed attempts, to update the rubrics. However, because generated games vary widely in objectives and mechanics, a single flat rubric set either remains too general (Jonsson & Svingby, 2007) or grows overly complex as new criteria accumulate. We therefore organize each stage's rubrics in a *two-level hierarchy* that groups individual rubrics under broad categories (Fig. 3, right). The PM retrieves and updates only the rubrics relevant to each game, so validation experience accumulates within each category instead of growing a monolithic rubric.

4 RTSGameAgent

Deploying VLM agents at BAR's scale poses two challenges. First, *unit action* requires per-unit decisions that grow with the number of combat units, while sparse VLM calls limit how often unit behavior is revised. Second, long-horizon play requires integrating recent events with historical context (Park et al., 2023; Packer et al., 2023). To address both, we introduce RTSGameAgent, a baseline agent harness (Fig. 4) combining group-level control with short- and long-term memory.

Scalable group-level control. To scale control with the number of combat units, we decompose *unit action* into two group-level action types, *group assignment* and *group command* (Fig. 4). With group assignment, the VLM dynamically organizes combat units into groups according to the current strategy and game state, unlike prior approaches that rely on predefined combat groups (Li et al.,

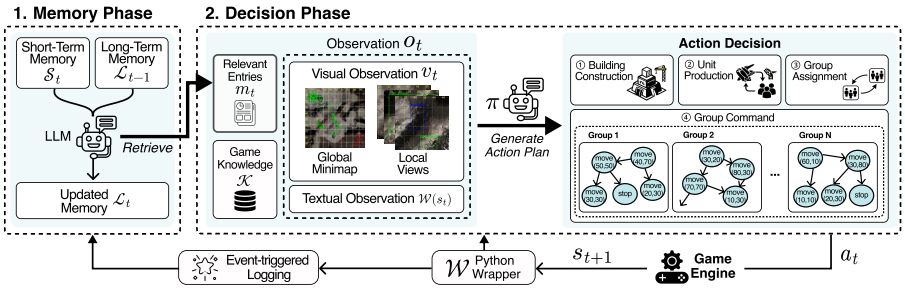


Figure 4: **Inference loop of RTSGAMEAGENT.** The *memory phase* (left) consolidates short-term logs $\mathcal{S}_t$ and long-term memory $\mathcal{L}_{t-1}$ into context m_t and updated memory $\mathcal{L}_t$; the *decision phase* (right) feeds m_t, observation o_t, and game knowledge $\mathcal{K}$ to the VLM policy π to output action a_t.

2025). For each group, a *group command* specifies one of three modes, `move`, `move_force`, or `stop`, together with a target coordinate when needed. Upon enemy contact, a group automatically engages (`fight`) and then resumes its command, except under `move_force`, which continues toward the target. Together, group assignment and group commands enable the VLM to express diverse group-level maneuvers within a compact control space.

To bridge sparse VLM calls, we further introduce *trajectory planning* over group commands. Instead of a single command, the VLM outputs for each group a condition-dependent command trajectory, represented as a directed acyclic graph (DAG), to sustain adaptive execution between calls. Transitions between nodes are triggered by predefined conditions on the group's own execution state; when a condition is satisfied, the corresponding follow-up command is activated without another VLM call. Because transitions are group-local, changes beyond a group's own state, including other groups' progress, are handled only at VLM calls. For critical changes requiring an immediate response, predefined triggers therefore initiate *replanning* before the next scheduled call, invoking the VLM with the updated game state to generate new trajectories. Detailed transition conditions and replanning triggers are provided in Appendix F.2.

Memory-augmented inference. To integrate recent events with historical context, we augment each inference with short- and long-term memory, following their cognitive distinction (Atkinson & Shiffrin, 1968; Squire, 1992). *Short-term memory* $\mathcal{S}_t$ holds event-triggered logs collected between VLM calls, such as enemy sightings within allied line of sight and battle events, whereas *long-term memory* $\mathcal{L}_t$ persists throughout the game, summarizing game progress and past battles, such as the resulting changes in allied and enemy forces. At each VLM call t, a memory phase and a decision phase run in sequence (Fig. 4). In the *memory phase*, the policy VLM, prompted with text only, consolidates $\mathcal{S}_t$ and $\mathcal{L}_{t-1}$ into $\mathcal{L}_t$ and extracts decision-relevant context m_t, *i.e.*, $(m_t, \mathcal{L}_t) = \mathrm{VLM}(\mathcal{S}_t, \mathcal{L}_{t-1})$, after which $\mathcal{S}_t$ is cleared. In the *decision phase*, the VLM receives m_t with the observation o_t, whose local camera views are centered on strategically relevant regions selected by heuristics. The base policy in Eq. 1 thus becomes $a_t = \pi(o_t, m_t \mid \mathcal{K})$, where a_t comprises building construction, unit production, group assignment, and group-command trajectories. Appendix F.3 details the inference loop, camera-view heuristics, and event-logging conditions.

5 EXPERIMENTS

Evaluation protocol. On full- and mini-games (Tabs. 3 and 4), we evaluate three agent baselines (Ma et al., 2024; Shao et al., 2024; Li et al., 2025) adapted to BAR (Appendix F.4), proprietary and open-weight VLMs, and a human baseline, all under the same interface and observations. Following Li et al. (2025), we use three difficulty levels (Lv.1–3; Details in Appendices D and E). Each setting is run 30 times (10 for humans), and Elo is computed from 20 round-robin matches per pair. The inference interval is 1 min of game time for full games, TCP, and FBA, and 3 s for MFD, ATS, and AWD (Appendix F.5). Self-evolving games are generated with Gemini-3.8-Flash.

Metrics. We primarily use win rate (WR) for full- and mini-game evaluation. As WR reflects only the final outcome, we additionally report six metrics in four dimensions to analyze how agents play: *resources* (gathered, RG; spent, RS), *army* (strength, AS; unit quality, UQ), *battle* (enemies destroyed, ED), and *visibility* (vision coverage, VC). Each metric is the agent's share relative to other players, scaled so that 0.5 denotes an even share. AS measures the total value of combat units

Table 3: **Full-game evaluation.** WR at built-in AIs (Lv.1–3). The evaluated agent, or a human using the same interface, occupies one slot; others are built-in AIs. Best in **bold**; second best underlined.

Model	*Duel (1v1)*			*Symmetric Team (2v2)*			*Asymm. Team (3v4)*			*Free-for-All (1v1v1v1)*		
	Lv.1	Lv.2	Lv.3	Lv.1	Lv.2	Lv.3	Lv.1	Lv.2	Lv.3	Lv.1	Lv.2	Lv.3
Agent Comparison (Backbone = Gemini-3.7-Flash)												
CoS (Ma et al., 2024)	0.17	0.00	0.00	0.20	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00
LLM-PySC2 (Li et al., 2025)	0.43	0.00	0.00	0.23	0.13	0.10	0.00	0.00	0.00	0.00	0.00	0.00
SwarmBrain (Shao et al., 2024)	0.80	0.23	0.13	0.57	0.33	**0.20**	0.00	0.00	0.00	0.00	0.00	0.00
RTSGAMEAGENT (**Ours**)	**1.00**	**0.43**	**0.30**	**0.83**	**0.40**	**0.20**	**0.17**	0.00	0.00	**0.20**	**0.07**	0.00
VLM Backbone Comparison (Agent = RTSGAMEAGENT*)*												
GPT-5.6-Luna	0.20	0.00	0.00	0.13	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Qwen3.8-27B	0.23	0.00	0.00	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Qwen3.5-397B	0.47	0.00	0.00	0.10	0.07	0.03	0.00	0.00	0.00	0.00	0.00	0.00
Muse-Spark-1.2	0.40	0.00	0.00	0.27	0.20	0.07	0.00	0.00	0.00	0.00	0.00	0.00
GPT-5.6-Terra	0.57	0.07	0.00	0.30	0.03	0.07	0.00	0.00	0.00	0.13	**0.07**	0.00
GLM-5.3-Flash	0.67	0.13	0.10	0.40	0.13	0.07	0.00	0.00	0.00	0.03	0.03	0.00
Grok-4.6	0.73	0.00	0.00	**0.83**	0.10	0.00	0.00	0.00	0.00	0.07	0.00	0.00
Gemini-3.7-Flash	**1.00**	**0.43**	**0.30**	**0.83**	**0.40**	**0.20**	**0.17**	0.00	0.00	**0.20**	**0.07**	0.00
Human	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.80	1.00	1.00	0.70

Table 4: **Mini-game evaluation.** WR over 30 runs per setting across Lv.1–3. CoS is evaluated only on TCP, as its heuristic army-wide action lacks the spatial unit control required by the other mini-games. Best in **bold**; second best underlined.

Model	**TCP**			**MFD**			**FBA**			**ATS**			**AWD**		
	Lv.1	Lv.2	Lv.3	Lv.1	Lv.2	Lv.3	Lv.1	Lv.2	Lv.3	Lv.1	Lv.2	Lv.3	Lv.1	Lv.2	Lv.3
Agent Comparison (Backbone = Gemini-3.7-Flash)															
CoS (Ma et al., 2024)	0.63	0.10	0.00	–	–	–	–	–	–	–	–	–	–	–	–
LLM-PySC2 (Li et al., 2025)	0.93	0.43	0.03	0.13	0.10	0.00	0.73	0.10	0.00	0.37	0.30	0.10	0.27	0.00	0.00
SwarmBrain (Shao et al., 2024)	**0.97**	0.40	0.07	0.67	0.00	0.00	**1.00**	0.33	**0.30**	0.87	**0.43**	**0.13**	0.10	0.00	0.00
RTSGAMEAGENT (**Ours**)	0.93	**0.67**	**0.20**	**0.93**	**0.60**	**0.37**	0.90	**0.47**	0.20	**1.00**	0.40	**0.13**	**0.77**	**0.43**	**0.13**
VLM Backbone Comparison (Agent = RTSGAMEAGENT*)*															
GPT-5.6-Luna	0.47	0.07	0.00	0.50	0.10	0.03	0.83	0.40	0.00	0.50	0.27	0.00	0.00	0.00	0.00
Qwen3.8-27B	0.60	0.00	0.00	0.63	0.13	0.00	0.83	0.03	0.10	0.20	0.17	0.00	0.20	0.03	0.00
Qwen3.5-397B	0.53	0.17	0.00	0.67	0.37	0.13	0.87	0.20	0.10	0.27	0.13	0.07	0.03	0.00	0.00
Muse-Spark-1.2	**1.00**	0.13	0.07	0.73	**0.60**	0.00	0.80	0.20	0.00	0.27	0.17	0.10	0.23	0.10	0.00
GPT-5.6-Terra	0.73	0.20	0.00	0.37	0.23	0.17	0.90	0.37	0.33	0.30	0.20	0.07	0.13	0.00	0.00
GLM-5.3-Flash	0.93	0.77	0.07	0.83	0.47	0.00	**1.00**	0.37	**0.40**	0.53	0.30	0.00	0.27	0.00	0.00
Grok-4.6	**1.00**	**0.83**	**0.23**	0.80	0.13	0.00	**1.00**	**0.57**	0.30	0.47	0.23	0.07	0.23	0.10	0.00
Gemini-3.7-Flash	0.93	0.67	0.20	**0.93**	**0.60**	**0.37**	0.90	0.47	0.20	**1.00**	**0.40**	**0.13**	**0.77**	**0.43**	**0.13**
Human	1.00	1.00	0.60	1.00	1.00	0.60	1.00	1.00	0.70	1.00	0.80	0.70	1.00	0.80	0.60

an agent maintains, whereas UQ measures the average value per produced combat unit, reflecting how advanced its composition is (Appendix G). To evaluate generated games (Sec.5.3), we use four criteria, detailed in Appendix H.2: *playability*, the fraction of generation attempts that yield a launchable game; *behavioral fidelity*, the fraction of expected behaviors, derived from the query and GDD, that are observed in play; *token usage* per generation; and pairwise *human preference*.

5.1 RESULTS

Full game evaluation. Tab. 3 reports full-game results (90% confidence intervals in Appendix D.2). For both agents and backbones, WR declines as the built-in AI level increases and drops sharply in the most interaction-demanding match-ups. Gemini-3.7-Flash matches or exceeds all other backbones in every setting, with Grok-4.6 and GLM-5.3-Flash competitive only in selected settings, so we use it for the agent comparison, which shows similar trends with other backbones (Appendix Tab. 11). RTSGAMEAGENT matches or outperforms the baselines in every setting and is the only agent to win any Asymmetric Team or Free-for-All game, supporting the benefit of its components (Sec.5.2). Nevertheless, no VLM agent exceeds 0.20 WR in these match-ups, even at Lv.1, where success requires tight ally coordination under numerical disadvantage and reasoning over multiple opponents, respectively. Humans under the same interface reach at least 0.70, indicating limitations of current VLM agents rather than infeasible settings.

Diagnostic mini-game evaluation. Tab. 4 reports mini-game results (90% confidence intervals in Appendix E.7). Among agents, RTSGAMEAGENT ranks first or second in every setting, with the largest margins on MFD and AWD; as both require committing forces against successive attacks, we conjecture that these margins stem from group-level control and the components built on it, the

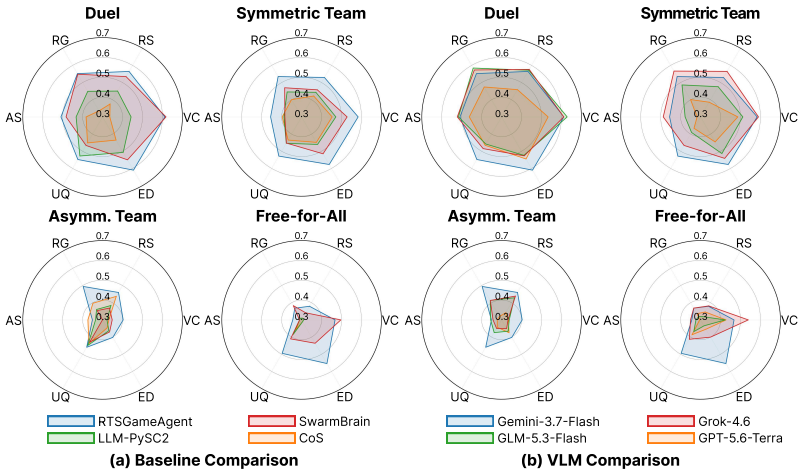


Figure 5: **Metric analysis of full-game performance.** Evaluated against Lv.1 built-in AIs.

Table 5: **Head-to-head Elo** from duels among Gemini-3.7-Flash-based agents.

Agent	Elo
CoS	1307
LLM-PySC2	1501
SwarmBrain	1562
RTSGameAgent	**1628**

Table 6: **Ablation study of RTSGameAgent.** We ablate each component and the visual observation, and report WR and detailed gameplay metrics in full-game duels; higher is better for all metrics.

Variant	WR ↑	RG ↑	RS ↑	AS ↑	UQ ↑	ED ↑	VC ↑
RTSGameAgent	**0.43**	0.39	**0.44**	**0.44**	**0.58**	**0.49**	0.33
w/o Group-Level Control	0.07	0.33	0.38	0.38	0.57	0.30	0.29
w/o Trajectory Planning	0.13	0.34	0.39	0.40	**0.58**	0.37	0.27
w/o Replanning	0.33	**0.40**	**0.44**	0.43	0.57	0.45	**0.34**
w/o Memory	0.20	0.37	0.42	0.40	**0.58**	0.34	0.30
w/o Visual Observation	0.20	0.37	0.38	0.39	0.57	0.39	0.20

most critical component in our ablation (Sec.5.2). Across backbones, although Gemini-3.7-Flash leads in full games, Grok-4.6 and GLM-5.3-Flash mostly match or exceed it on TCP and FBA but fall behind on MFD, ATS, and AWD at every level. We conjecture that they are at least as strong at economic throughput and scouting, central to TCP and FBA, but weaker at handling forces as engagements unfold, central to the other three, a difference less visible in full-game results. ATS results are also consistent with those in Symmetric Team full games (Tab. 3), which likewise require allied coordination, with Gemini-3.7-Flash best or tied for best in both at every level; while not a one-to-one measure of collaboration, this agreement suggests that ATS captures behavior relevant to coordination. Humans under the same interface achieve near-perfect WR at Lv.1–2 and 0.60–0.70 at Lv.3, whereas no model exceeds 0.23 on TCP, ATS, and AWD at Lv.3; the largest gaps, on ATS and AWD, suggest that collaboration and anticipatory planning may be common weaknesses of current VLM agents, consistent with their failures in team full games.

5.2 Detailed Analyses

Metric analysis. Fig. 5 reports additional metrics for (a) agents under Gemini-3.7-Flash and (b) the four strong backbones in full games (Tab. 3). We measure them at Lv.1, where most models win some games. SwarmBrain nearly matches RTSGameAgent in RG, RS, and VC in Duel, and Grok-4.6 matches or exceeds Gemini-3.7-Flash in RG, RS, AS, and VC in Duel and Symmetric Team, yet both score lower in UQ and ED. We conjecture that, beyond economy and vision, winning requires fielding stronger units and committing them to battle. The latter is consistent with the largest mini-game gains of RTSGameAgent and Gemini-3.7-Flash on MFD and AWD, both of which require committing forces against successive attacks. In Asymmetric Team and Free-for-All, where WR barely separates models, nearly all shares fall below 0.5 and AS drops to about 0.3 for every model, suggesting that agents fail to sustain combat forces against multiple opponents.

Comparing agents. To compare agents, we perform head-to-head round-robin duels, whose Elo ratings (Tab. 5) match the built-in-AI ranking. In particular, RTSGameAgent wins 14 of 20 duels against SwarmBrain (Shao et al., 2024), the strongest baseline, with a 90% Wilson interval above 0.5. This agreement supports built-in AIs as a consistent reference for ranking agents.

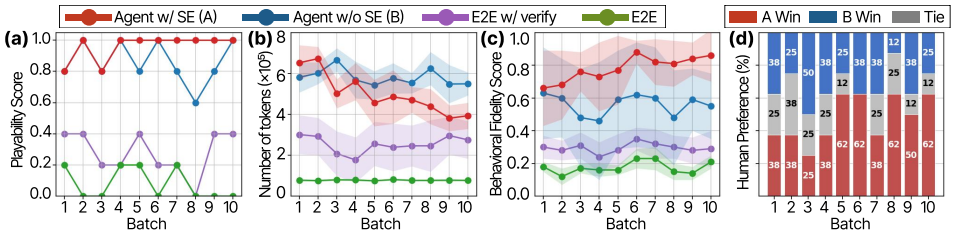


Figure 6: **Self-evolution over successive generation.** (a) Playability, (b) Token Usage, (c) Behavioral Fidelity, and (d) Human Preference between A and B across ten batches of five generations.

Ablation study. We ablate each component of RTSGAMEAGENT and visual observation in full-game duels against Lv.2 built-in AIs, whose intermediate difficulty lets each ablation reveal its effect (Tab. 6). Removing *group-level control*, which also disables trajectory planning and replanning built on group commands, causes the largest drops in WR and ED, mainly hurting coordination in engagements. Removing *trajectory planning* lowers WR, ED, and VC, suggesting that condition-dependent follow-up commands sustain execution and map control between sparse VLM calls. Without *replanning*, resource, army, and vision metrics stay comparable or even increase, yet ED and WR decrease; we conjecture that it helps respond to critical events in time rather than build strength. Removing *memory* lowers WR and ED with modest changes elsewhere, suggesting that temporal context turns similar strength into more effective engagements. Overall, ED varies most among the auxiliary metrics and largely follows WR, whereas UQ stays nearly constant, indicating that the components mainly improve how forces are committed rather than which units are fielded (*cf*. Fig. 5). Removing *vision*, *i.e.*, dropping v_t from o_t, reduces VC the most and also lowers ED and WR, suggesting that visual observations help maintain map control, which in turn supports engagements.

5.3 EVALUATING GENERATED GAMES

Fig. 6 plots the four criteria as experience accumulates over ten sequential batches of five queries. We compare the multi-agent pipeline with and without self-evolution (SE), *Agent w/ SE* (A) and *Agent w/o SE* (B), with an end-to-end baseline that generates the entire game in one pass, with and without iterative re-verification (*E2E w/ verify* and *E2E*; Appendix H.2.5). Appendix H.4 shows an example generation cycle, rule and rubric updates, and representative generated games, and Appendix H.5 reports a preliminary agent evaluation on generated games.

Multi-agent pipeline *vs*. end-to-end generation. Both Agent variants yield launchable games in nearly every generation and achieve far higher behavioral fidelity than the E2E variants, even without self-evolution. The E2E variants consume fewer tokens only by skipping stage-wise design. Repeated verification improves E2E only modestly (*E2E w/ verify*), so it cannot replace this design.

Effect of self-evolution. Self-evolution improves quality and efficiency. Over batches, the behavioral fidelity of *Agent w/ SE* rises and its token usage falls, whereas *Agent w/o SE* shows no such trend. *Agent w/ SE* also reaches full playability after a few batches, whereas *Agent w/o SE* yields unlaunchable games in some batches. Human preference between the Agent variants, the only ones with high playability, shifts toward *Agent w/ SE* in later batches. Removing either database reuse or rubric updates weakens the fidelity and token trends (Appendix H.3), showing that each contributes.

6 CONCLUSION

We propose RTSGAMEBENCH, a benchmark built on *Beyond All Reason* that provides holistic evaluation via full games across diverse match-up structures, diagnostic assessment via mini-games each emphasizing a focal strategic demand, and extensible coverage via a self-evolving game generation framework. For VLMs to operate in large-scale RTS games, we also introduce RTSGAMEAGENT, a baseline agent harness combining group-level trajectory planning with short- and long-term memory. Using RTSGAMEAGENT, we evaluate SoTA VLMs and find that their performance declines as built-in AIs strengthen and falls short of humans, particularly when they must coordinate with allies or contend with multiple opponents. Mini-games further suggest that the relative strengths of VLMs vary across strategic demands. In game generation, experience-driven evolution improves game quality while reducing token usage over successive cycles.

AI USE STATEMENT

We utilized AI solely to polish the writing, such as improving the grammar, phrasing, and readability of author-written drafts. All AI-suggested edits were reviewed and revised by the authors.

REFERENCES

Daechul Ahn, San Kim, and Jonghyun Choi. Society of mind meets real-time strategy: A hierarchical multi-agent framework for strategic reasoning. In *Conference on Language Modeling (COLM)*, 2025.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems (NeurIPS)*, pp. 23716–23736, 2022.

Timothée Anne, Noah Syrkis, Meriem Elhosni, Florian Turati, Franck Legendre, Alain Jaquier, and Sebastian Risi. Harnessing language for coordination: A framework and benchmark for llm-driven multiagent control. *IEEE Transactions on Games*, 17(4):933–943, 2025.

Richard C Atkinson and Richard M Shiffrin. Human memory: A proposed system and its control processes. In *Psychology of Learning and Motivation*, volume 2, pp. 89–195. Academic Press, 1968.

Beyond All Reason Developers. Beyond all reason: Main game repository. `https://github.com/beyond-all-reason/Beyond-All-Reason`, 2019.

Beyond All Reason Developers. Recoil engine: A powerful free cross-platform RTS game engine. `https://github.com/beyond-all-reason/RecoilEngine`, 2023. A hard fork of the SpringRTS engine (version 105 tree).

Beyond All Reason Team. Beyond all reason. `https://www.beyondallreason.info/`, 2024. Open-source real-time strategy game.

Michael Buro. Real-time strategy games: A new ai research challenge. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1534–1535, 2003.

Dake Chen, Haoyang Zhang, Hanbin Wang, Yunhao Huo, Yuzhao Li, and Junjie Wang. GameGPT: Multi-agent collaborative framework for game development. *arXiv preprint arXiv:2310.08067*, 2023.

Maxence Faldor, Jenny Zhang, Antoine Cully, and Jeff Clune. Omni-epic: Open-endedness via models of human notions of interestingness with environments programmed in code. In *International Conference on Learning Representations (ICLR)*, pp. 85260–85385, 2025.

Fandom Contributors. StarCraft II — StarCraft wiki. `https://starcraft.fandom.com/wiki/StarCraft_II`, 2024. Accessed: 2025.

Tracy Fullerton. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. CRC Press, Boca Raton, FL, 3rd edition, 2014.

Kanishk Gandhi, Dorsa Sadigh, and Noah D Goodman. Strategic reasoning with language models. *arXiv preprint arXiv:2305.19165*, 2023.

Mark Hendrikx, Sebastiaan Meijer, Joeri van der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 9(1):1:1–1:22, 2013. doi: 10.1145/2422956.2422957.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, pp. 23247–23275, 2024.

Lanxiang Hu, Mingjia Huo, Yuxuan Zhang, Haoyang Yu, Eric P Xing, Ion Stoica, Tajana Rosing, Haojian Jin, and Hao Zhang. lmgame-bench: How good are llms at playing games? In *International Conference on Learning Representations (ICLR)*, pp. 81308–81356, 2026.

Sihao Hu, Tiansheng Huang, and Ling Liu. Pokéllmon: A human-parity agent for pokémon battles with large language models. *arXiv preprint arXiv:2402.01118*, 2024.

Roberto Ierusalimschy. *Programming in Lua*. Lua.org, 2006.

Anders Jonsson and Gunilla Svingby. The use of scoring rubrics: Reliability, validity and educational consequences. *Educational Research Review*, 2(2):130–144, 2007. doi: 10.1016/j.edurev.2007.05.002.

Mohd Hozaifa Khan and Ravi Kiran Sarvadevabhatla. Sketchtopia: A dataset and foundational agents for benchmarking asynchronous multimodal communication with iconic feedback. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18176–18186, 2025.

San Kim, Daechul Ahn, and Jonghyun Choi. A multimodal RTS game benchmark for evaluating vision-language models. *Journal of KIISE*, 53(4):331–338, 2026. doi: 10.5626/JOK.2026.53.4.331.

Klaus Krippendorff. Computing Krippendorff's alpha-reliability. Departmental Papers (ASC) 43, Annenberg School for Communication, University of Pennsylvania, 2011. URL `https://repository.upenn.edu/asc_papers/43`.

Jiacheng Li, Jiahui Liu, Yuqing Wang, Gaochen Cui, Xiao Zhang, Qianchuan Zhao, Ziyou Zhang, and Chenghao Li. From winning to understanding: A diagnostic long-horizon RTS benchmark for LLMs. In *International Conference on Machine Learning (ICML)*, 2026.

Zongyuan Li, Yanan Ni, Runnan Qi, Chang Lu, Lumin Jiang, Xu Xiaojie, Xiangbei Liu, Pengfei Li, Yunzheng Guo, Zhe Ma, et al. Llm-pysc2: Starcraft ii learning environment for large language models. *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 127595–127657, 2025.

Jonathan Light, Min Cai, Sheng Shen, and Ziniu Hu. Avalonbench: Evaluating llms playing the game of avalon. *arXiv preprint arXiv:2310.05036*, 2023.

Mingxian Lin, Shengju Qian, Yuqi Liu, Yi-Hua Huang, Yiyu Wang, Wei Huang, Yitang Li, Fan Zhang, Zeyu Hu, Lingting Zhu, Xin Wang, and Xiaojuan Qi. OmniGameArena: A unified UE5 benchmark for VLM game agents with improvement dynamics. *arXiv preprint arXiv:2606.09826*, 2026.

Wenye Lin, Jonathan Roberts, Yunhan Yang, Samuel Albanie, Zongqing Lu, and Kai Han. Gamebot: Transparent assessment of llm reasoning in games. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 7656–7682, 2025.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics (TACL)*, pp. 157–173, 2024.

Qian Long, Zhi Li, Ran Gong, Ying Nian Wu, Demetri Terzopoulos, and Xiaofeng Gao. Teamcraft: A benchmark for multi-modal multi-agent systems in minecraft. *arXiv preprint arXiv:2412.05255*, 2024.

Weiyu Ma, Qirui Mi, Yongcheng Zeng, Xue Yan, Yuqiao Wu, Runji Lin, Haifeng Zhang, and Jun Wang. Large language models play starcraft ii: Benchmarks and a chain of summarization approach. *Advances in neural information processing systems (NeurIPS)*, pp. 133386–133442, 2024.

Weiyu Ma, Yuqian Fu, Zecheng Zhang, Bernard Ghanem, and Guohao Li. Ava: Attentive vlm agent for mastering starcraft ii. In *Findings of the Association for Computational Linguistics (ACL)*, pp. 4270–4290, 2026.

Santiago Ontañón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(4):293–311, 2013.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wołczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, et al. Balrog: Benchmarking agentic llm and vlm reasoning on games. In *International Conference on Learning Representations (ICLR)*, pp. 96666–96702, 2025.

Dongmin Park, Minkyu Kim, Beongjun Choi, Junhyuck Kim, Keon Lee, Jonghyun Lee, Inkyu Park, ByeongUk Lee, Jaeyoung Hwang, Jaewoo Ahn, et al. Orak: A foundational benchmark for training and evaluating llm agents on diverse video games. In *International Conference on Learning Representations (ICLR)*, pp. 60684–60744, 2026.

Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *ACM Symposium on User Interface Software and Technology (UIST)*, pp. 1–22, 2023.

Siyuan Qi, Shuo Chen, Yexin Li, Xiangyu Kong, Junqi Wang, Bangcheng Yang, Yifan Zhong, Xiaoyuan Zhang, Zhaowei Zhang, Nian Liu, et al. Civrealm: A learning and reasoning odyssey in civilization for decision-making agents. In *International Conference on Learning Representations (ICLR)*, pp. 27995–28026, 2024.

Qualitipedia contributors. Beyond all reason. `https://newqualitipedia.telepedia.net/wiki/Beyond_All_Reason`, 2024.

Glen Robertson and Ian Watson. A review of real-time strategy game ai. *AI Magazine*, 35(4): 75–104, 2014.

Xiao Shao, Weifu Jiang, Fei Zuo, and Mengqing Liu. Swarmbrain: Embodied agent for real-time strategy game starcraft ii via large language models. *arXiv preprint arXiv:2401.17749*, 2024.

Pengbo Shen, Yaqing Wang, Ni Mu, Yao Luan, Runpeng Xie, Senhao Yang, Lexiang Wang, Hao Hu, Shuang Xu, Yiqin Yang, et al. Sc2arena and starevolve: Benchmark and self-improvement framework for llms in complex decision-making tasks. *arXiv preprint arXiv:2508.10428*, 2025.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International conference on machine learning (ICML)*, pp. 31210–31227, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems (NeurIPS)*, pp. 8634–8652, 2023.

Larry R Squire. Memory and the hippocampus: A synthesis from findings with rats, monkeys, and humans. *Psychological Review*, 99(2):195–231, 1992.

Wenjie Tang, Yuan Zhou, Keyan Cheng, Erqiang Xu, Liquan Xiao, and Minne Li. Dsgbench: A diverse strategic game benchmark for evaluating llm-based agents in complex decision-making environments. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 16987–16991, 2026.

Vivek Verma, David Huang, William Chen, Dan Klein, and Nicholas Tomlin. Measuring general intelligence with generated games. *arXiv preprint arXiv:2505.07215*, 2025.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Dawei Wang, Chengming Zhou, Di Zhao, Xinyuan Liu, Marci Chi Ma, Gary Ushaw, and Richard Davison. Towermind: A tower defence game learning environment and benchmark for llm as agents. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 26151–26159, 2026.

Ziyue Wang, Yurui Dong, Fuwen Luo, Minyuan Ruan, Zhili Cheng, Chi Chen, Peng Li, and Yang Liu. How do multimodal large language models handle complex multimodal reasoning? placing them in an extensible escape game. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4807–4817, 2025.

Zelai Xu, Zhexuan Xu, Xiangmin Yi, Huining Yuan, Mo Guang, Kaiwen Long, Xinlei Chen, Yi Wu, Chao Yu, and Yu Wang. Vs-bench: Evaluating vlms for strategic abilities in multi-agent environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21561–21572, 2026.

Jianzhu Yao, Kevin Wang, Ryan Hsieh, Haisu Zhou, Tianqing Zou, Zerui Cheng, Zhangyang Wang, and Pramod Viswanath. SPIN-Bench: How well do LLMs plan strategically and reason socially? *arXiv preprint arXiv:2503.12349*, 2025.

Jiayi Zhang, Yiran Peng, Fanqi Kong, Cheng Yang, Yifan Wu, Zhaoyang Yu, Jinyu Xiang, Jianhao Ruan, Jinlin Wang, Maojia Song, HongZhang Liu, Xiangru Tang, Bang Liu, Chenglin Wu, and Yuyu Luo. Autoenv: Automated environments for measuring cross-environment agent learning. *arXiv preprint arXiv:2511.19304*, 2025.

Yadong Zhang, Shaoguang Mao, Tao Ge, Xun Wang, Adrian de Wynter, Yan Xia, Wenshan Wu, Ting Song, Man Lan, and Furu Wei. Llm as a mastermind: A survey of strategic reasoning with large language models. In *Conference on Language Modeling (COLM)*, 2024.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 19632–19642, 2024.

Xiangxi Zheng, Linjie Li, Zhengyuan Yang, Ping Yu, Alex Jinpeng Wang, Rui Yan, Yuan Yao, and Lijuan Wang. V-mage: A game evaluation framework for assessing vision-centric capabilities in multimodal large language models. In *Findings of the Association for Computational Linguistics (ACL)*, pp. 17707–17758, 2026.

Xinyue Zheng, Haowei Lin, Kaichen He, Zihao Wang, Qiang Fu, Haobo Fu, Zilong Zheng, and Yitao Liang. Mcu: An evaluation framework for open-ended game agents. In *International Conference on Machine Learning (ICML)*, 2025.

APPENDIX CONTENTS

A Beyond All Reason: Gameplay and Scalability **15**

A.1 Game Characteristics for Strategic Reasoning 15

A.2 Scale Comparison with StarCraft II 15

B Further Comparison with Existing Game Benchmarks for LLM and VLM Agents **17**

C Game Interface and Static Game Knowledge $\mathcal{K}$ **19**

C.1 Game Interface 19

C.2 Static Game Knowledge $\mathcal{K}$ 20

D Details of Full Games **21**

D.1 Evaluation Setup 21

D.2 Results with Confidence Intervals 23

D.3 Agent Comparison with Other Backbones 23

E Details of Diagnostic Mini-Games **23**

E.1 Human Ratings of Strategic Demands 24

E.2 Time-Constrained Production (TCP) — Resource Management 25

E.3 Multi-Front Defense (MFD) — Spatial & Temporal Reasoning 26

E.4 Fortified Base Assault (FBA) — Opponent Modeling 26

E.5 Allied Team Skirmish (ATS) — Collaboration 27

E.6 Anticipatory Wave Defense (AWD) — Adversarial Planning 29

E.7 Results with Confidence Intervals 29

F Detailed Procedure of RTSGAMEAGENT **29**

F.1 Action Space 29

F.2 Trajectory Planning and Replanning 30

F.3 Inference Loop 31

F.4 Adapting StarCraft II LLM Agents to BAR 32

F.5 Inference Interval Analysis 33

G Details about Evaluation Metrics for RTSGAMEBENCH **34**

H Details of the Self-Evolving Game Generation Framework **35**

H.1 Algorithm Details 35

H.2 Evaluation of Generated Games 37

H.3 Ablation of Self-Evolution Mechanisms 41

H.4 Examples of Self-Evolving Game Generation 41

H.5 Agent Evaluation on Generated Games 42

A BEYOND ALL REASON: GAMEPLAY AND SCALABILITY

A.1 GAME CHARACTERISTICS FOR STRATEGIC REASONING

Beyond All Reason (BAR) (Beyond All Reason Team, 2024) is an open-source real-time strategy (RTS) game built on the Recoil engine (Beyond All Reason Developers, 2023), a fork of SpringRTS. BAR supports substantially larger battles than StarCraft II (SC2) in unit count, player count, and battlefield size (Sec. A.2). Players choose from three factions, **Armada**, **Cortex**, and **Legion**, each with a distinct playstyle. Armada emphasizes flexible and mobile warfare with cloaking and precision weapons, Cortex favors durable and defensive play with heavily armored units, and Legion adopts an aggressive style centered on resource-intensive offensive units. In our full games, all players use Armada (Sec. D).

Resource management. BAR uses two primary resources, *metal* and *energy*, whose production and consumption must be kept in balance. Metal is extracted from map deposits or reclaimed from wreckage, whereas energy is generated by structures such as solar collectors, wind turbines, geothermal power plants, and fusion reactors. BAR employs a *streaming economy*, in which resources are generated and consumed as continuous flows rather than managed solely as finite stockpiles. Production throughput therefore depends on resource income, requiring players to continuously trade off economic expansion, technological progression, and immediate military investment.

Base construction and technology progression. BAR enforces explicit construction and technology dependencies, making advanced units and structures available only after their prerequisites are completed. Effective production therefore requires sequential reasoning over build orders, technology paths, and the allocation of construction capacity across multiple facilities.

Low-level automation. BAR delegates much of the per-unit and economic execution to the game engine, reducing routine micro-management (Beyond All Reason Team, 2024)[2]. In combat, armed units automatically select and fire at enemies within range according to their firing stance (fire at will, return fire, or hold fire), while their movement state (hold position, maneuver, or roam) determines how far they pursue targets. Given a destination, units plan their own routes through engine-level pathfinding. An attack-move makes them stop to engage enemies within firing range before continuing, whereas a plain move prioritizes reaching the destination. On the economic side, construction units execute queued build orders in sequence, assist other constructors and factories, and reclaim or repair everything within a designated area. Factories can repeat their production queues, and energy converters run only when stored energy exceeds a set threshold. Consequently, a single command, such as sending a group to a position, expands into many low-level actions without further input. These mechanisms automate how commands are executed, not which commands to issue. RTSGAMEAGENT's group commands build on them, with `move` engaging enemies on contact before resuming and `move_force` continuing toward the target (Sec. 4).

Army coordination and strategic maneuvering. Effective play therefore centers on coordinating forces at the group level. Players must decide how to organize units into groups and allocate them across fronts, when and where to engage, and how to maneuver them as battlefield conditions evolve under fog-of-war. These coordination and planning demands are central to the strategic capabilities that RTSGAMEBENCH is designed to evaluate.

A.2 SCALE COMPARISON WITH STARCRAFT II

Unit variety. SC2 and BAR each feature three playable factions. SC2 contains 96 unique units and buildings across Terran, Protoss, and Zerg, whereas BAR contains 554 (Beyond All Reason Developers, 2019). BAR further supports a multi-tier progression system with faction-specific units across Tier 1, Tier 2, and Tier 3, as well as multiple movement domains spanning land, air, sea, and amphibious operations. This broader unit space increases the diversity of possible army compositions, counters, and strategic interactions.

Supply capacity. SC2 uses a *weighted* population system in which units consume different amounts of supply (Fandom Contributors, 2024). For example, a Battlecruiser consumes six supply, whereas a Marine consumes one. Thus, the standard per-player cap of 200 supply does not correspond to

[2] `https://www.beyondallreason.info/commands-20`

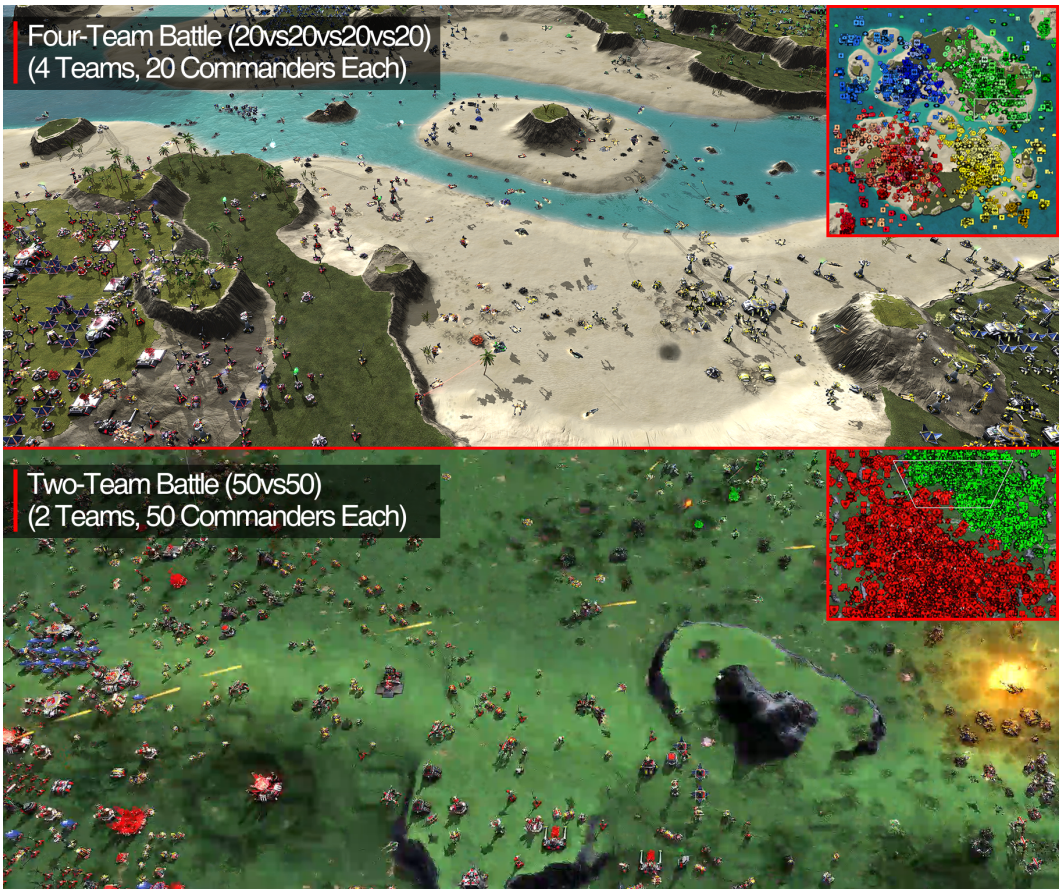


Figure 7: **Large-scale combat environments in BAR. Top.** A four-team battle with 20 commanders per team (80 in total). **Bottom.** A 50 *vs*. 50 team battle with 50 commanders per side. Such battles involve large numbers of players and thousands of units within a single battlefield.

200 fielded units. A late-game army with approximately 60 workers, for instance, typically leaves room for roughly 110–130 combat units, depending on army composition. For this reason, the SC2 supply cap in Tab. 1 is marked with an asterisk, indicating that nominal supply should not be read directly as unit count. BAR instead counts each unit equally toward its unit limit regardless of tier or combat strength, with a default per-player cap of 2,000 units (Beyond All Reason Developers, 2019). This limit is configurable, and we use the default in our experiments.

Unit capacity. Multiplying the eight-player limit by the 200-supply cap gives SC2 a nominal upper bound of 1,600 supply. Because supply is weighted by unit type and a large fraction typically goes to workers, this value overstates the number of simultaneously fielded combat units; the illustrative late-game range above yields approximately 880–1,040 combat units across eight players. In contrast, BAR supports up to 32,000 simultaneous units across all players (Beyond All Reason Developers, 2019).

Map size. We compare the largest available maps in each game. The SC2 map editor limits playable maps to 256×256 tiles (Fandom Contributors, 2024). BAR specifies map dimensions in map units of 512 elmos, the engine's distance unit, each spanning 64 squares of 8 elmos. The largest official BAR map measures 32×32 map units, *i.e.*, $2{,}048 \times 2{,}048$ squares. Treating an SC2 tile and a BAR square as comparable grid cells, the largest BAR map covers approximately $64\times$ the area of the largest SC2 map (Beyond All Reason Team, 2024).

Player limit. Standard SC2 multiplayer supports up to eight players (Fandom Contributors, 2024). BAR officially supports up to 100 players in a single match (Beyond All Reason Team, 2024), although competitive matches commonly use smaller configurations such as 8v8.

Table 7: **Comparison with game-based benchmarks for LLM/VLM agents.** Single-game benchmarks are listed separately from multi-game suites, for which a property is marked if at least one constituent game or evaluation track supports it; in contrast, RTSGAMEBENCH provides all properties within a single environment. *Diagnostic Evaluation* indicates evaluation that decomposes performance by specific demands or abilities beyond aggregate outcomes, via targeted scenarios or dimension-wise scores. *Scenario Generation* indicates automated generation of new evaluation scenarios, *e.g.*, tasks or games specified by an LLM (✔), procedural or manual variation of scenarios within predefined task families (△), or a fixed scenario set (✘).

Benchmark	Game(s)	Imperfect Information	Multi-Agent	Multimodal Input	Diagnostic Evaluation	Scenario Generation
Single-game benchmarks						
EscapeCraft (Wang et al., 2025)	Room Escape Game	✘	✘	✔	✘	△
AvalonBench (Light et al., 2023)	Avalon	✔	✔	✘	✘	✘
Sketchtopia (Khan & Sarvadevabhatla, 2025)	Sketchtopia	✔	✔	✔	✘	✘
CivRealm (Qi et al., 2024)	FreeCiv	✔	✔	✘	✔	✘
TeamCraft (Long et al., 2024)	Minecraft	✔	✔	✔	✘	△
MCU (Zheng et al., 2025)	Minecraft	✔	✘	✔	✔	✔
Multi-game suites						
SPIN-Bench (Yao et al., 2025)	Multiple	✔	✔	✘	✔	✘
DSGBench (Tang et al., 2026)	Multiple	✔	✔	✘	✔	△
gg-bench (Verma et al., 2025)	Generated	✔	✔	✘	✘	✔
VS-Bench (Xu et al., 2026)	Multiple	✔	✔	✔	✔	✘
OmniGameArena (Lin et al., 2026)	Multiple	✔	✔	✔	✔	✘
RTSGAMEBENCH (Ours)	Beyond All Reason	✔	✔	✔	✔	✔

Large-scale engagements in practice. Tab. 1 reports the maximum scale supported by each game, whereas Fig. 7 shows how this capacity translates into large multiplayer battles, where coordination complexity grows with the numbers of players, units, and interacting fronts. This scale motivates our use of BAR as a testbed for strategic reasoning under large-scale multi-agent interaction.

B FURTHER COMPARISON WITH EXISTING GAME BENCHMARKS FOR LLM AND VLM AGENTS

We compare against game-based benchmarks for LLM and VLM agents that target strategic or multi-agent decision-making or support scenario generation (Tab. 7), and against the RTS benchmarks for LLM and VLM agents that we are aware of (Tab. 8). The first two properties in Tab. 7 follow from our definition of strategic reasoning, *i.e.*, deciding under uncertainty (imperfect information) while interacting with other agents (multi-agent), and the remaining three from our focus on VLMs (multimodal input) and the diagnostic and extensible desiderata in Sec. 1 (diagnostic evaluation and scenario generation).

Game-based benchmarks. As shown in Tab. 7, single-game benchmarks cover only subsets of the five properties, and multi-game suites broaden coverage by combining games with complementary properties. MCU (Zheng et al., 2025) generates tasks with an LLM and scores trajectories along multiple dimensions, but targets single-agent play in Minecraft. VS-Bench (Xu et al., 2026) is the closest in scope, evaluating VLMs in cooperative, competitive, and mixed-motive environments and separating strategic reasoning, measured by next-action prediction, from decision-making; its environments, however, are a fixed set of small-scale board, card, and video games. DSGBench (Tang et al., 2026) scores LLM agents along five capability dimensions across six text-based strategic games, including *StarCraft II*, and supports manual customization of scenario variables such as opponent strategies. SPIN-Bench (Yao et al., 2025) varies action spaces, state complexity, and the number of interacting agents across planning, board, card, and negotiation tasks, relying on textual observations. OmniGameArena (Lin et al., 2026) evaluates VLM agents in newly built solo, competitive, and cooperative games, each designed around a subset of embodied capabilities such as spatial navigation and reaction, and focuses on how performance improves through reflection. gg-bench (Verma et al., 2025) is the closest to our generation framework, using an LLM to generate new two-player turn-based games as a data-generating benchmark; its games are text-based, and its pipeline samples novel games rather than targeting strategic demands specified by users. Among the compared benchmarks, only RTSGAMEBENCH fully supports all five properties, and it does so within a single large-scale RTS environment.

Table 8: **Comparison with RTS-specific benchmarks.** *Full-Game Match-ups* lists the player configurations of full-game evaluation (–: scenario-based evaluation only; Solo: a single player against scripted enemy waves); match-ups refer to player-level configurations rather than the number of LLM agents controlling a single player. *Diagnostic Scenarios* indicates controlled scenarios designed around specific strategic demands or abilities (△: controlled variations not organized around strategic demands, *e.g.*, opponent-strategy or rule perturbations); unlike *Diagnostic Evaluation* in Tab. 7, it does not count dimension-wise scores, which fall under *Behavioral Metrics*. *Behavioral Metrics* indicates in-game statistics that characterize gameplay beyond task outcome or its graded progress, *e.g.*, resource, combat, or visibility statistics. *Scenario Generation* follows Tab. 7. Build, Prod., and Action denote building construction, unit production, and unit actions, respectively. †The *StarCraft II* component of a multi-game suite.

Benchmark	Game	Multimodal Input	Full-Game Match-ups	Diagnostic Scenarios	Behavioral Metrics	Scenario Generation	Action Space Categories
HIVE (Anne et al., 2025)	Hand-Crafted Game	✘	–	✔	✘	✘	Action
TowerMind (Wang et al., 2026)	Hand-Crafted Game	✔	Solo	✘	✘	△	Build + Prod. + Action
TextStarCraftII (Ma et al., 2024)	StarCraft II	✘	1v1	✘	✔	✘	Build + Prod.
TextSCII-All (Ahn et al., 2025)	StarCraft II	✘	1v1	✘	✘	✘	Build + Prod.
SC2Arena (Shen et al., 2025)	StarCraft II	✘	1v1	✘	✔	✘	Build + Prod. + Action
DSGBench (Tang et al., 2026)	StarCraft II†	✘	1v1	△	✔	△	Build + Prod.
AVACraft (Ma et al., 2026)	StarCraft II	✔	–	✔	✘	✘	Action
LLM-PySC2 (Li et al., 2025)	StarCraft II	✔	1v1	✔	✔	✘	Build + Prod. + Action
Kim et al. (Kim et al., 2026)	StarCraft II	✔	1v1	✘	✘	✘	Build + Prod. + Action
Li et al. (Li et al., 2026)	Red Alert	✘	1v1	△	✔	✘	Build + Prod. + Action
RTSGAMEBENCH (Ours)	Beyond All Reason (BAR)	✔	1v1, Team, FFA	✔	✔	✔	Build + Prod. + Action

RTS-specific benchmarks. Prior RTS benchmarks evaluate agents in simplified hand-crafted environments, in *StarCraft II*, or, more recently, in *Red Alert* (Tab. 8). Several track behavioral metrics beyond win rate (Ma et al., 2024; Shen et al., 2025; Tang et al., 2026; Li et al., 2025; 2026). The *Red Alert* benchmark of Li et al. (2026) logs economy, combat-loss, and visibility statistics and ranks models by head-to-head Elo ratings, in the same spirit as our auxiliary metrics and head-to-head evaluation (Sec. 5.2). It differs from RTSGAMEBENCH in that agents receive structured textual states, play one-on-one matches against rule-based opponents of distinct playstyles or other LLMs, and are diagnosed through opponent-style and rule variations and language interventions rather than scenarios organized around strategic demands. Unlike these one-on-one metrics, our auxiliary metrics are designed for multi-player match-ups, normalizing per-player quantities by alive span and reporting odds-normalized shares (Sec. G). Some benchmarks also probe specific abilities through controlled scenarios (Anne et al., 2025; Ma et al., 2026; Li et al., 2025). HIVE and AVACraft do not pair these scenarios with full-game evaluation, and LLM-PySC2 pairs its mini-scenarios only with one-on-one games. None of them organizes its scenarios around a taxonomy of RTS strategic demands whose alignment is verified by human ratings (Sec. E.1). Overall, no prior RTS benchmark in Tab. 8 combines the following properties, and none provides (ii) or (iv): (i) multimodal observations that support direct evaluation of VLM agents; (ii) full games across diverse match-up structures, including team and free-for-all settings; (iii) diagnostic mini-games grounded in a taxonomy of RTS AI challenges (Buro, 2003), each emphasizing a focal strategic demand; (iv) automated generation of new diagnostic scenarios from free-form user queries; and (v) an action space spanning building construction, unit production, and unit action. Moreover, BAR provides substantially larger battles, maps, and player configurations than *StarCraft II* (Tab. 1; Sec. A.2), enabling evaluation under larger-scale coordination and longer-horizon strategic demands.

Automated game and environment generation. LLMs have also been used to generate games and environments, mostly for purposes other than diagnostic evaluation. GameGPT (Chen et al., 2023) automates game development with a role-specialized multi-agent framework; OMNI-EPIC (Faldor et al., 2025) and AutoEnv (Zhang et al., 2025) generate executable environments in code to drive open-ended learning and to study agent learning across heterogeneous environments, respectively; and gg-bench (Verma et al., 2025) generates games to benchmark general reasoning. Our framework similarly adopts role-specialized agents (Hong et al., 2024; Chen et al., 2023), but targets diagnostic evaluation: it generates scenarios on demand from free-form queries within a complex RTS engine, designs each around a focal strategic demand, and validates it through staged rubric checks and simulation tests. Moreover, whereas task archives condition the proposal of novel tasks (Faldor et al., 2025), our framework reuses validated design documents and rule code to bypass redundant stages and accumulates validation experience in hierarchical rubrics (Sec. 3.3).

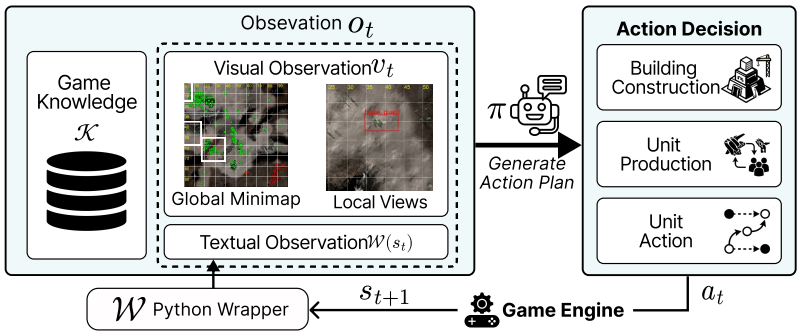


Figure 8: **Overview of the base agent interface.** At each decision step t, the agent receives a multimodal observation o_t consisting of visual observations v_t (a global minimap and up to four local camera views) and a structured textual observation $\mathcal{W}(s_t)$ extracted from the game state by a Python wrapper. Together with static game knowledge $\mathcal{K}$, the VLM policy π generates an action a_t over three action categories: building construction, unit production, and unit action. The action is executed by the game environment, producing the next state s_{t+1} and continuing the observe–decide–act loop.

C Game Interface and Static Game Knowledge $\mathcal{K}$

Fig. 8 illustrates the base agent interface shared across all evaluation settings in RTSGameBench. This section describes the interface (Sec. C.1), covering the observe–decide–act loop, the observation space, and the action space, followed by the static game knowledge $\mathcal{K}$ provided to the agent (Sec. C.2).

C.1 Game Interface

Observe–decide–act loop. Every scenario in RTSGameBench follows a common observe–decide–act loop. Before each game begins, the agent receives static game knowledge $\mathcal{K}$ (Sec. C.2), which contains the scenario description, available units and buildings, and team configuration. At each decision step t, the environment constructs a multimodal observation

$$o_t = \big(v_t,\ \mathcal{W}(s_t)\big), \tag{2}$$

where v_t denotes the visual observations and $\mathcal{W}(s_t)$ denotes the structured textual observation extracted from the engine state s_t by a Python wrapper $\mathcal{W}$. The agent policy π, instantiated by a VLM, then selects an action conditioned on the current observation and static game knowledge:

$$a_t = \pi(o_t \mid \mathcal{K}), \qquad s_{t+1} \leftarrow \mathrm{Env}(s_t, a_t). \tag{3}$$

This process repeats at fixed decision intervals, and the environment pauses during model inference so that evaluation reflects strategic decision quality rather than inference latency or reaction speed.

Visual observation v_t. The visual input comprises one *global minimap* and up to four *local camera views*. The minimap provides a bird's-eye overview of the battlefield, while the local views provide higher-resolution observations of strategically relevant regions selected per decision step. Candidate camera locations are determined according to the following priority order: (i) locations of recent combat events, (ii) the commander's position when its health falls below 50%, (iii) visible enemy force clusters, (iv) allied unit clusters, (v) the commander's position when it is not in danger, and (vi) the allied home base. Candidates are processed in this order, and spatially similar locations are merged to avoid redundant observations. The four highest-priority distinct locations are then selected as local camera views, or fewer when fewer than four distinct candidates are available. When Fog-of-War (FoW) is enabled, both the minimap and the local views are restricted to information available within allied line of sight, which prevents the visual observations from revealing hidden enemy activity and introduces partial observability into the agent's decision process.

Textual observation $\mathcal{W}(s_t)$. The Python wrapper $\mathcal{W}$ converts the engine state s_t into a structured textual representation suitable for the VLM. It includes metal and energy levels; the type, position,

Table 9: **Summary of the unit and building encyclopedia in $\mathcal{K}$.** The table reports the number of Armada entities in each category and technology tier.

Category	T1	T2	T3	Total
Bot	9	17	4	30
Vehicle	10	12	1	23
Air	13	10	0	23
Sea	7	12	0	19
Hover	5	0	1	6
Factory	7	5	1	13
Defense	17	12	0	29
Building	18	16	0	34
Total	**86**	**84**	**7**	**177**

and health of allied and currently visible enemy units; ongoing construction and production queues; and the current game time. The two channels are therefore complementary, as the visual channels capture spatial battlefield structure while the textual channel provides structured game-state information for decision-making.

Action space. The base action space consists of three categories:

- **Building construction**: the agent selects a constructor, a building type, and a target location.
- **Unit production**: the agent selects a production building and a unit type to produce.
- **Unit action**: the agent selects one or more units and specifies an action and, when required, a target location.

All spatial locations are represented on a normalized $(0, 0)$–$(100, 100)$ coordinate grid. Once an action is issued, the game engine handles low-level execution, including pathfinding, collision avoidance, and other unit-level mechanics.

C.2 Static Game Knowledge $\mathcal{K}$

The static game knowledge $\mathcal{K}$ is provided to the agent once before each game and remains fixed throughout the episode. It consists of three components: (i) a scenario description specifying the task objective and win/loss conditions, (ii) a team configuration specifying allied and enemy factions, and (iii) a unit and building encyclopedia describing the entities available to the agent.

All experiments in this work use the *Armada* faction for the evaluated agent. Accordingly, the encyclopedia in $\mathcal{K}$ contains only Armada-side entities. The opposing *Cortex* faction may appear as an enemy in the evaluated scenarios, but its unit and building encyclopedia is not included in $\mathcal{K}$.

Unit and building encyclopedia. The encyclopedia catalogs the entities available to the Armada faction and organizes them into eight categories: *Bot*, *Vehicle*, *Air*, *Sea*, *Hover*, *Factory*, *Defense*, and *Building*, where *Building* includes economy and utility structures. Each entry contains the entity name, internal engine code, technology tier (T1, T2, or T3), and a natural-language description of its role and capabilities. For constructor units and production buildings, each entry provides a set of *build options* specifying the structures or units that the entity can construct or produce.

Technology tiers. Units and buildings are organized into three technology tiers corresponding to BAR's progression system:

- **T1 (Tech 1)**: Basic units and structures available during the early stage of the game. They are relatively inexpensive and quick to produce, forming the foundation of early-game armies and economies (*e.g.*, Pawn infantry bot, Solar Collector, and Bot Lab).
- **T2 (Tech 2)**: Advanced units and structures unlocked through higher-tier factories and constructors. They are generally more capable and specialized but require greater resource investment (*e.g.*, Sharpshooter sniper bot, Fusion Reactor, and Advanced Bot Lab).

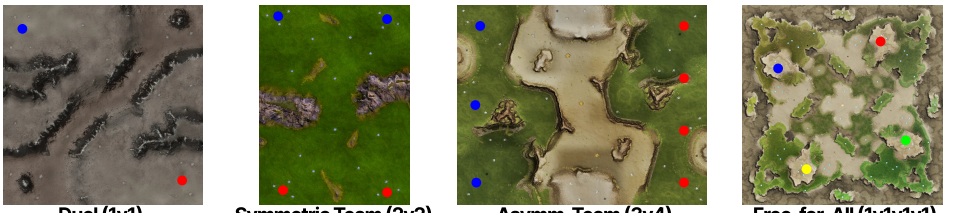


Figure 9: **Map layouts and starting positions.** Blue markers indicate the starting positions of the agent's team, including allied AIs, and markers in other colors indicate enemy starting positions.

- **T3 (Tech 3)**: Experimental units produced through the Experimental Gantry. These units provide substantial combat capability at high resource cost (*e.g.*, Titan, Thor, and Razorback).

Category coverage. Table 9 summarizes the number of entities in each category and technology tier. The encyclopedia contains 177 Armada entities in total, spanning combat units, constructors, factories, defensive structures, and economy and utility buildings. The complete encyclopedia, including natural-language descriptions and build options, is provided to the agent as part of $\mathcal{K}$.

Factories and production chains. A key component of the encyclopedia is the production dependency structure encoded through each entity's build options. Each factory specifies the units it can produce. For example, the Bot Lab (T1) produces basic bot units such as Pawn, Tick, and Mace, whereas the Advanced Bot Lab (T2) provides access to more advanced units such as Gunslinger, Sharpshooter, and Fatboy. At the highest tier, the Experimental Gantry (T3) produces six experimental units: Vanguard, Titan, Marauder, and Razorback as bots, Thor as a vehicle, and Lunkhead as a hover unit.

Constructor units similarly specify the buildings and structures they can construct. For example, the Construction Bot and Advanced Construction Vehicle provide access to different portions of the building technology tree. Together, these factory and constructor dependencies define the production and technology progression that the agent must navigate when planning economic development, unit production, and technological advancement.

D Details of Full Games

D.1 Evaluation Setup

Faction configuration. All players in every matchup use the Armada faction. Although the environment also supports Cortex, using a single faction removes inter-faction asymmetries, so that performance differences reflect agent behavior rather than faction-specific advantages.

Built-in AI difficulty. All slots other than the evaluated agent, both allied and enemy, are filled by BAR's built-in AI, BARbarian (Beyond All Reason Developers, 2019). RTSGameBench defines three difficulty levels, Lv.1, Lv.2, and Lv.3, which correspond to its Easy, Normal, and Hard settings, respectively.

Map configurations. Each matchup is played on a fixed map (Fig. 9), with map sizes given in BAR's standard map-size units. The 1v1 matchup uses *BarR 1.1* (8×8), the 2v2 matchup uses *Pinewood Derby v1* (12×6), the 3v4 matchup uses *Wanderlust 2.1* (10×8), and the 1v1v1v1 matchup uses *Center Command BAR v1.0* (16×8).

Prompts and examples. Figures 15, 16, and 17 show the system prompt, an input example, and an output example for the memory phase, respectively. Figures 18, 19, and 20 show the corresponding prompt and examples for the decision phase.

Qualitative example. Figure 10 shows a mid-game moment from a 1v1 full game, in which the agent expands its economy while preparing for upcoming combat.

Table 10: **Full-game evaluation results with 90% Wilson score confidence intervals.** Each cell reports the win rate over 30 runs; the upper limit of the interval is shown in red superscript and the lower limit in green subscript. Human results are omitted since ten runs yield intervals too wide to be informative.

Model	**Duel** (1v1)	**Sym. Team** (2v2)	**Asymm. Team** (3v4)	**FFA** (1v1v1v1)
Lv.1				
CoS (Ma et al., 2024)	$0.17^{0.30}_{0.08}$	$0.20^{0.34}_{0.11}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
LLM-PySC2 (Li et al., 2025)	$0.43^{0.58}_{0.30}$	$0.23^{0.38}_{0.13}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
SwarmBrain (Shao et al., 2024)	$0.80^{0.89}_{0.66}$	$0.57^{0.70}_{0.42}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
RTSGAMEAGENT (Ours)	$1.00^{1.00}_{0.92}$	$0.83^{0.92}_{0.70}$	$0.17^{0.30}_{0.08}$	$0.20^{0.34}_{0.11}$
GPT-5.6-Luna	$0.20^{0.34}_{0.11}$	$0.13^{0.27}_{0.06}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Qwen3.8-27B	$0.23^{0.38}_{0.13}$	$0.33^{0.48}_{0.21}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Qwen3.5-397B	$0.47^{0.61}_{0.33}$	$0.10^{0.23}_{0.04}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Muse-Spark-1.2	$0.40^{0.55}_{0.27}$	$0.27^{0.41}_{0.16}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
GPT-5.6-Terra	$0.57^{0.70}_{0.42}$	$0.30^{0.45}_{0.18}$	$0.00^{0.08}_{0.00}$	$0.13^{0.27}_{0.06}$
GLM-5.3-Flash	$0.67^{0.79}_{0.52}$	$0.40^{0.55}_{0.27}$	$0.00^{0.08}_{0.00}$	$0.03^{0.14}_{0.01}$
Grok-4.6	$0.73^{0.84}_{0.59}$	$0.83^{0.92}_{0.70}$	$0.00^{0.08}_{0.00}$	$0.07^{0.18}_{0.02}$
Gemini-3.7-Flash	$1.00^{1.00}_{0.92}$	$0.83^{0.92}_{0.70}$	$0.17^{0.30}_{0.08}$	$0.20^{0.34}_{0.11}$
Lv.2				
CoS (Ma et al., 2024)	$0.00^{0.08}_{0.00}$	$0.07^{0.18}_{0.02}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
LLM-PySC2 (Li et al., 2025)	$0.00^{0.08}_{0.00}$	$0.13^{0.27}_{0.06}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
SwarmBrain (Shao et al., 2024)	$0.23^{0.38}_{0.13}$	$0.33^{0.48}_{0.21}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
RTSGAMEAGENT (Ours)	$0.43^{0.58}_{0.30}$	$0.40^{0.55}_{0.27}$	$0.00^{0.08}_{0.00}$	$0.07^{0.18}_{0.02}$
GPT-5.6-Luna	$0.00^{0.08}_{0.00}$	$0.10^{0.23}_{0.04}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Qwen3.8-27B	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Qwen3.5-397B	$0.00^{0.08}_{0.00}$	$0.07^{0.18}_{0.02}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Muse-Spark-1.2	$0.00^{0.08}_{0.00}$	$0.20^{0.34}_{0.11}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
GPT-5.6-Terra	$0.07^{0.18}_{0.02}$	$0.03^{0.14}_{0.01}$	$0.00^{0.08}_{0.00}$	$0.07^{0.18}_{0.02}$
GLM-5.3-Flash	$0.13^{0.27}_{0.06}$	$0.13^{0.27}_{0.06}$	$0.00^{0.08}_{0.00}$	$0.03^{0.14}_{0.01}$
Grok-4.6	$0.00^{0.08}_{0.00}$	$0.10^{0.23}_{0.04}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Gemini-3.7-Flash	$0.43^{0.58}_{0.30}$	$0.40^{0.55}_{0.27}$	$0.00^{0.08}_{0.00}$	$0.07^{0.18}_{0.02}$
Lv.3				
CoS (Ma et al., 2024)	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
LLM-PySC2 (Li et al., 2025)	$0.00^{0.08}_{0.00}$	$0.10^{0.23}_{0.04}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
SwarmBrain (Shao et al., 2024)	$0.13^{0.27}_{0.06}$	$0.20^{0.34}_{0.11}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
RTSGAMEAGENT (Ours)	$0.30^{0.45}_{0.18}$	$0.20^{0.34}_{0.11}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
GPT-5.6-Luna	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Qwen3.8-27B	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Qwen3.5-397B	$0.00^{0.08}_{0.00}$	$0.03^{0.14}_{0.01}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Muse-Spark-1.2	$0.00^{0.08}_{0.00}$	$0.07^{0.18}_{0.02}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
GPT-5.6-Terra	$0.00^{0.08}_{0.00}$	$0.07^{0.18}_{0.02}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
GLM-5.3-Flash	$0.10^{0.23}_{0.04}$	$0.07^{0.18}_{0.02}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Grok-4.6	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Gemini-3.7-Flash	$0.30^{0.45}_{0.18}$	$0.20^{0.34}_{0.11}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$

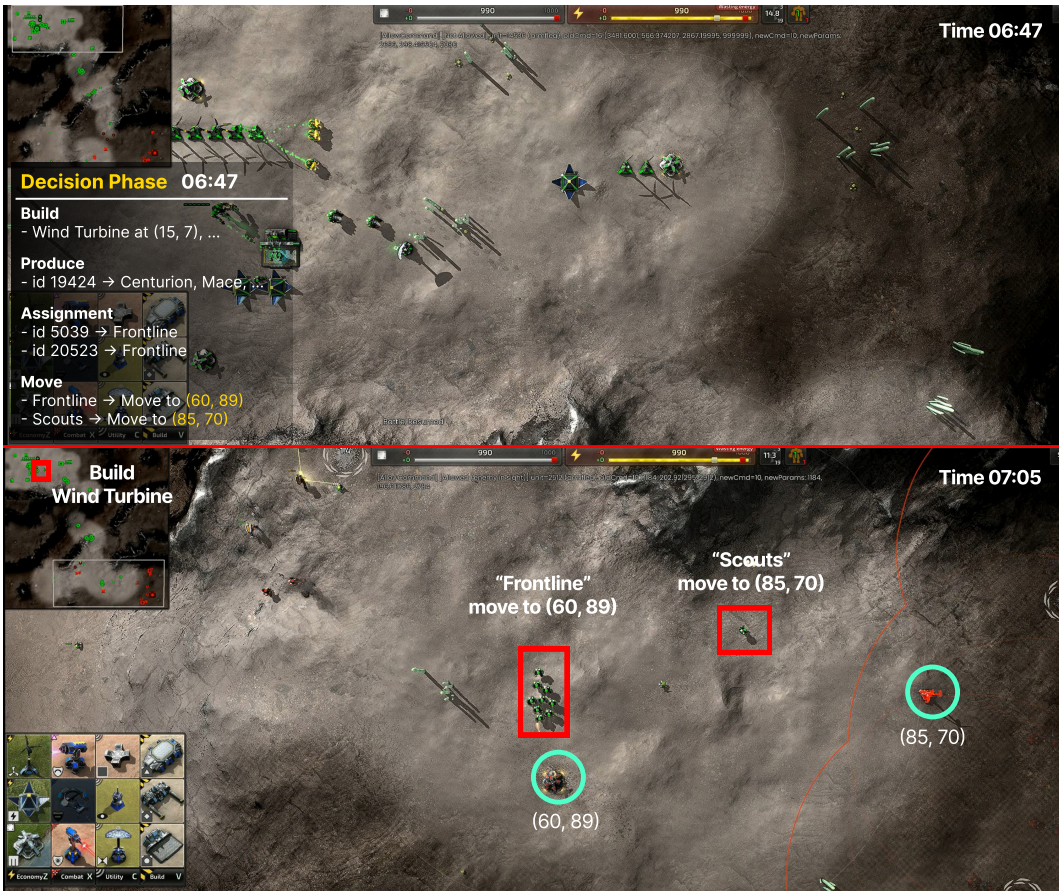


Figure 10: **Example of decision execution in a full game.** At 06:47 (top), the agent issues high-level decisions, including building construction, unit production, and group movement commands with target coordinates. At 07:05 (bottom), the frontline units advance toward $(60, 89)$ and the scouts move toward $(85, 70)$, confirming that the issued commands are executed in game.

D.2 Results with Confidence Intervals

Tab. 10 repeats the full-game win rates of Tab. 3 together with 90% Wilson score confidence intervals over 30 runs per setting. A win rate of zero over 30 runs still admits a true win rate of up to about 8%, so the many zero entries at higher difficulty levels indicate that wins are rare rather than impossible.

D.3 Agent Comparison with Other Backbones

The agent comparison in Tab. 3 uses Gemini-3.7-Flash, which matches or exceeds all other backbones. To check whether the comparison depends on this choice, we repeat it with GPT-5.6-Terra and GLM-5.3-Flash under the same protocol (Tab. 11). The trends remain similar, although absolute win rates are lower. As with Gemini-3.7-Flash, WR declines as the built-in AI level increases and stays low in Asymmetric Team and Free-for-All. RTSGameAgent also matches or outperforms the baselines in nearly every setting, except Symmetric Team Lv.1 and Lv.2 with GPT-5.6-Terra.

E Details of Diagnostic Mini-Games

We design five diagnostic mini-games, each targeting a distinct strategic demand from a taxonomy of RTS AI challenges (Buro, 2003): *resource management* (RM), *spatial & temporal reasoning* (S&T), *opponent modeling* (OM), *collaboration* (COL), and *adversarial planning* (AP). Each mini-game

Table 11: **Agent comparison with other backbones.** WR of each agent with GPT-5.6-Terra and GLM-5.3-Flash as backbones, under the same protocol as Tab. 3 (30 runs per setting). Best and second-best non-zero results within each backbone are shown in **bold** and underlined, respectively; ties share the same formatting.

Model	*Duel* (1v1)			*Symmetric Team* (2v2)			*Asymm. Team* (3v4)			*Free-for-All* (1v1v1v1)		
	Lv.1	Lv.2	Lv.3	Lv.1	Lv.2	Lv.3	Lv.1	Lv.2	Lv.3	Lv.1	Lv.2	Lv.3
Backbone = GPT-5.6-Terra												
CoS (Ma et al., 2024)	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
LLM-PySC2 (Li et al., 2025)	0.13	0.00	0.00	**0.37**	**0.07**	0.00	0.00	0.00	0.00	0.03	0.00	0.00
SwarmBrain (Shao et al., 2024)	0.10	0.00	0.00	0.10	0.00	0.00	0.00	0.00	0.00	0.07	0.00	0.00
RTSGAMEAGENT (Ours)	**0.57**	**0.07**	0.00	0.30	0.03	**0.07**	0.00	0.00	0.00	**0.13**	**0.07**	0.00
Backbone = GLM-5.3-Flash												
CoS (Ma et al., 2024)	0.00	0.00	0.00	0.10	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00
LLM-PySC2 (Li et al., 2025)	0.07	0.00	0.00	0.37	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00
SwarmBrain (Shao et al., 2024)	0.53	0.03	0.00	0.37	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00
RTSGAMEAGENT (Ours)	**0.67**	**0.13**	**0.10**	**0.40**	**0.13**	**0.07**	0.00	0.00	0.00	**0.03**	**0.03**	0.00

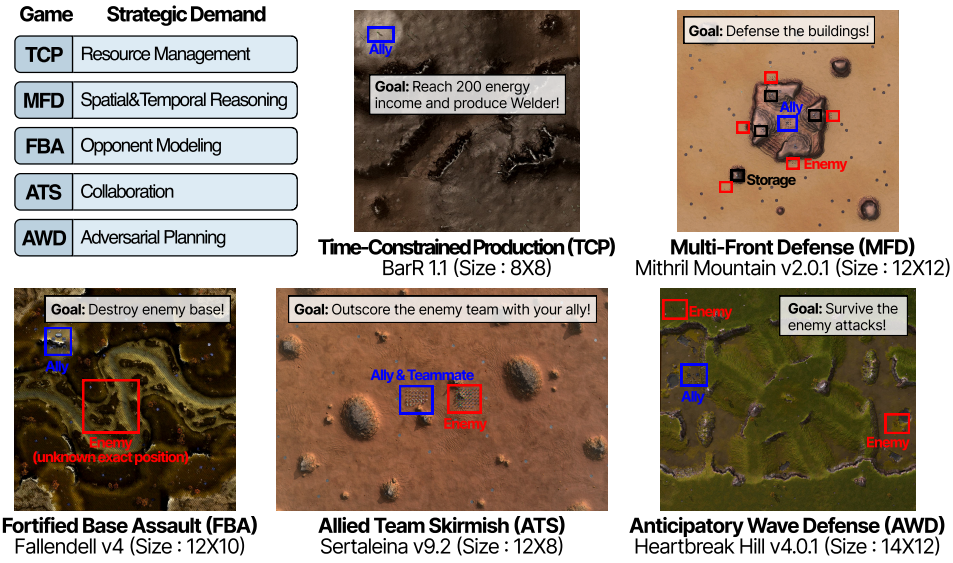


Figure 11: **Initial configurations of the diagnostic mini-games.** The figure illustrates the map layouts and initial unit placements for the five diagnostic mini-games used in our evaluation. Blue markers indicate the controlled agent and its teammates, while red markers denote enemy units or enemy base locations. In scenarios where the exact enemy position is unknown, the red box indicates the region where the enemy may appear.

emphasizes its focal demand through the objective, the available action types, and the information made visible to the agent, and we describe below what that demand amounts to in BAR. Their map layouts and initial configurations are illustrated in Fig. 11. A sixth demand, *decision making under uncertainty*, is incorporated selectively through Fog-of-War (FoW) rather than as a separate mini-game. The taxonomy also includes pathfinding, which we do not target, as BAR's engine resolves it automatically once a destination is issued (Sec. A).

Every mini-game is evaluated at three difficulty levels (Lv.1–3). Rather than changing the objective, each level scales the difficulty of achieving it, mostly by increasing the *power* of the enemy forces, BAR's internal value scale that converts energy into metal-equivalent units (Sec. G).

E.1 HUMAN RATINGS OF STRATEGIC DEMANDS

To verify that each mini-game emphasizes its intended strategic demand, we collect ratings from five human raters on how strongly every mini-game elicits each of the five focal demands targeted by the mini-games (Tab. 2), *i.e.*, RM, S&T, OM, COL, and AP.

Table 12: **Human ratings of strategic demands.** Mean ± standard deviation over five raters, on a 1–5 scale. Each mini-game's focal demand is in **bold**.

Mini-game	RM	S&T	OM	COL	AP
TCP	**4.8** ± 0.45	1.8 ± 0.84	1.2 ± 0.45	1.0 ± 0.00	2.4 ± 0.89
MFD	1.2 ± 0.45	**4.6** ± 0.55	2.0 ± 0.71	1.2 ± 0.45	3.4 ± 0.89
FBA	3.2 ± 0.84	3.0 ± 0.00	**4.2** ± 0.45	1.0 ± 0.00	3.0 ± 0.71
ATS	1.8 ± 0.45	3.6 ± 0.55	2.6 ± 0.89	**4.6** ± 0.55	2.6 ± 0.55
AWD	3.0 ± 0.71	3.2 ± 0.45	2.2 ± 0.45	1.0 ± 0.00	**4.4** ± 0.55

Demand definitions. Raters receive a brief definition of each demand, adapted from the taxonomy of RTS AI challenges (Buro, 2003) and identical across all scenarios. RM concerns gathering resources and allocating them among economy, production, and technology; S&T concerns reasoning about positions, terrain, and the timing of actions; OM concerns inferring the opponent's forces or strategy from in-game observations; COL concerns coordinating with allies whose actions the player does not control; and AP concerns planning ahead against the adversary's future actions. Whereas Buro (2003) frames opponent modeling as learning opponents' weaknesses across games, we define it as inference within an episode, since each mini-game is played as a single episode. We exclude pathfinding, which BAR handles at the engine level (Sec. A.1), and decision making under uncertainty, which we control through the FoW setting rather than scenario design (Tab. 2).

Protocol. The five raters are experienced RTS players familiar with *StarCraft II* who were not involved in designing the benchmark. To avoid revealing the intended focal demands, mini-games are presented in a randomized order under neutral labels (Scenario A–E), and raters are not informed that each mini-game targets a particular demand. For each scenario, a rater reads player-facing instructions that describe only the mechanics, *i.e.*, the initial setup, win/loss conditions, and available actions, without the design rationale given in Sec. 3.2. The rater then plays one episode at Lv.1 before rating, so that ratings reflect the demands encountered in actual play rather than the instructions alone. After playing, the rater scores all five demands on a 1–5 scale, where 1 indicates that the demand is irrelevant to winning and 5 that it is central. Raters score independently, without access to other raters' scores. Fig. 2 reports the mean over the five raters, and Tab. 12 additionally reports the standard deviation of each cell.

Alignment with focal demands. Every mini-game receives its highest mean rating on its focal demand, ranging from 4.2 to 4.8, and this also holds for each rater individually: in all 25 rater–mini-game pairs, the focal demand receives the highest score, strictly so in 19 and tied for highest in the remaining 6. Focal demands average 4.52 against 2.22 for the others, with per-mini-game margins of 1.0–2.4 over the next highest demand.

Inter-rater agreement. Raters largely agree with one another, with a mean pairwise correlation of 0.83 between raters' 25 ratings (five mini-games × five demands) and a per-cell standard deviation of 0.51 on average (at most 0.89).

Off-diagonal patterns. Off-diagonal ratings remain non-negligible, *e.g.*, ATS scores 3.6 on S&T and MFD scores 3.4 on AP, supporting that each mini-game emphasizes one demand while preserving the coupled decision-making of RTS gameplay rather than isolating a single capability. Collaboration is the most sharply separated demand, scoring 4.6 on ATS but at most 1.2 elsewhere, as ATS is the only mini-game with an ally. FBA and ATS show the smallest margins over the next highest demand (1.0 each), as assaulting a fortified base also draws on RM, S&T, and AP (3.0–3.2), and fighting alongside an ally also draws on S&T (3.6).

E.2 Time-Constrained Production (TCP) — Resource Management

Strategic demand. BAR's streaming economy generates and consumes metal and energy as continuous flows, so income, tech progression, and unit production constantly compete for the same budget (Sec. A). Construction and technology dependencies further make the order of these investments consequential, as an advanced unit becomes available only after its prerequisites are complete.

Objective. The agent must reach a metal income of at least 20 and an energy income of at least 200 as quickly as possible within a 10-minute limit, while fielding the units required by the difficulty level. A run counts as a win if all targets are met before the time limit. The available action categories are **Build**, **Prod.**, and **Action**, and FoW is **enabled**.

Difficulty levels. All levels share the same income targets and time limit, and differ in the units the agent must field. Lv.1 requires no units beyond the income targets, Lv.2 additionally requires at least three Pawns and two Stouts, and Lv.3 requires at least one Welder, a T2 unit that becomes available only after an additional tech step.

Design rationale. At Lv.2 and Lv.3, the income targets and the unit requirements pull resources in different directions. The former demand investment in economic structures, whereas the latter require production buildings and, at Lv.3, a further tech step before any unit can be produced. Because BAR's production dependencies make each stage contingent on earlier ones, inefficient early decisions delay the entire chain and consume the limited time budget.

Game description. At the beginning of each episode, the agent receives a natural-language description of the targets and the time constraint, *e.g.*, at Lv.3:

> *Reach a metal income of 20 and an energy income of 200, and field at least one Welder, as quickly as possible within 10 minutes.*

E.3 Multi-Front Defense (MFD) — Spatial & Temporal Reasoning

Strategic demand. BAR's large maps make travel time a first-class constraint, so the agent must decide not only where forces should be but also when they must start moving to arrive in time. Forces committed to one location are unavailable elsewhere until they redeploy, which couples spatial and temporal decisions.

Objective. The agent must defend a set of Energy Storage buildings located on a mountain for 30 seconds, during which enemy forces attack from different directions at staggered times. Losing any of the buildings results in defeat, so a run counts as a win only if all of them survive. The only available action category is **Action**, as building construction and unit production are disabled, and FoW is **disabled**.

Difficulty levels. Lv.1 and Lv.2 place five storage buildings, and Lv.2 increases the total power of the attacking forces by 50% over Lv.1. Lv.3 adds a sixth building and increases the attacking power by 100% over Lv.1, so that a larger enemy force is spread over one more front.

Design rationale. With a fixed force and no production, success depends entirely on where the forces are placed and when they are moved. The staggered attacks require the agent to reason jointly about spatial distance, travel time, and the temporal order of incoming threats. Forces committed to one building may have to be relocated before a later attack reaches another, while moving them too early leaves the current front exposed. Disabling FoW keeps the task focused on positioning and redeployment rather than on gathering information.

Game description. At the beginning of each episode, the agent receives a natural-language description of the objectives to defend and the defeat condition, *e.g.*, at Lv.1:

> *Defend the five Energy Storage buildings on the mountain for 30 seconds. Losing any of them results in defeat.*

E.4 Fortified Base Assault (FBA) — Opponent Modeling

Strategic demand. Under partial observability, the agent must infer the opponent's forces, defenses, and likely response from what it has observed, and adapt its own composition and approach accordingly. BAR's counter relationships among unit types make this inference consequential, as a force effective against one composition can be ineffective against another.

Objective. The agent must destroy the enemy Advanced Metal Storage, located inside a fortified base, within a 10-minute limit. A run counts as a win if the target is destroyed before the time limit. The available action categories are **Build**, **Prod.**, and **Action**, and FoW is **enabled**, so the defensive layout is not immediately visible.

Difficulty levels. Lv.1 uses the base configuration, while Lv.2 and Lv.3 increase the total power of the base by 100% and 200%, respectively, so that the agent must assemble a stronger force and choose its approach more carefully.

Design rationale. The target is protected by defensive structures and combat forces whose composition and placement determine which attack strategies succeed. Since the layout is revealed only as the agent scouts, it must infer the defensive configuration from partial observations and adapt its production, force composition, and approach accordingly. For example, concentrated anti-air defenses discourage an air assault, while a heavily fortified approach favors an alternative route. As further parts of the base become visible, the agent may need to revise its assessment and plan.

Game description. At the beginning of each episode, the agent receives a natural-language description of the target and the time constraint:

> *Destroy the enemy Advanced Metal Storage within 10 minutes.*

E.5 Allied Team Skirmish (ATS) — Collaboration

Strategic demand. Team play requires acting jointly with allies the agent does not control, which entails dividing fronts, producing forces that complement allied ones, and timing engagements together. Since outcomes are shared, decisions that benefit the agent alone do not necessarily benefit the team.

Objective. The agent fights alongside an ally in a 2v2 team battle for 30 seconds. The outcome is decided at the team level from the damage each team deals and the army strength it maintains over the episode. Let D denote the total damage, in hit points, that a team deals, and let AS denote its army strength, *i.e.*, the area under the curve of the total power of its surviving combat units over the episode, with power defined as in Sec. G. Unlike the auxiliary AS metric, which is normalized by each player's alive span (Sec. G), AS here is accumulated over the whole episode, so that units lost during the battle lower the team's share. With subscripts *own* and *opp* denoting the agent's team (the agent and its ally) and the opposing team, the two shares are

$$s_{\text{dmg}} = \frac{D_{\text{own}}}{D_{\text{own}} + D_{\text{opp}}}, \qquad s_{\text{AS}} = \frac{\text{AS}_{\text{own}}}{\text{AS}_{\text{own}} + \text{AS}_{\text{opp}}}, \tag{4}$$

and the team with the higher weighted sum $0.5\, s_{\text{dmg}} + 0.5\, s_{\text{AS}}$ wins. Since the two shares of the opposing teams sum to one, a run counts as a win when the agent's team exceeds 0.5, *i.e.*, when it inflicts more damage and maintains a stronger force than the opponents on average. The available action categories are **Prod.** and **Action**, and FoW is **disabled**.

Difficulty levels. The levels vary the production budget of the agent's team relative to the opposing team: $+10\%$ at Lv.1, equal budgets at Lv.2, and -10% at Lv.3. Since both teams fight with comparable forces, the outcome depends less on raw force and more on how well the agent coordinates with its ally.

Design rationale. The agent does not control its ally and must therefore account for the ally's behavior when deciding how to produce and move its own units. Effective collaboration may involve reinforcing an allied engagement, producing units that complement the allied composition, or covering regions the ally leaves undefended, whereas poor coordination leads to fragmented attacks and redundant force allocation. Since both shares are measured over the whole team, preserving the agent's own units at the expense of the ally does not improve the outcome. Disabling FoW lets the agent observe allied and enemy activity, so that the task measures coordination rather than information gathering.

Game description. At the beginning of each episode, the agent receives a natural-language description of the team structure, the cooperative objective, and how the outcome is evaluated:

> *Fight alongside your ally in a 2v2 team battle for 30 seconds. Your performance is evaluated from the damage your team deals and the forces it preserves.*

Table 13: **Mini-game evaluation results with 90% Wilson score confidence intervals.** Each cell reports the win rate over 30 runs; the upper limit of the interval is shown in red superscript and the lower limit in green subscript. We report Wilson score intervals rather than the normal approximation, which is unreliable for small samples and degenerates at win rates of 0 and 1. CoS is evaluated only on TCP, as its heuristic army-wide control lacks the spatial unit assignment required by the other mini-games.

Model	TCP	MFD	FBA	ATS	AWD
Lv.1					
CoS (Ma et al., 2024)	$0.63^{0.76}_{0.48}$	—	—	—	—
LLM-PySC2 (Li et al., 2025)	$0.93^{0.98}_{0.82}$	$0.13^{0.27}_{0.06}$	$0.73^{0.84}_{0.59}$	$0.37^{0.52}_{0.24}$	$0.27^{0.41}_{0.16}$
SwarmBrain (Shao et al., 2024)	$0.97^{0.99}_{0.86}$	$0.67^{0.79}_{0.52}$	$1.00^{1.00}_{0.92}$	$0.87^{0.94}_{0.73}$	$0.10^{0.23}_{0.04}$
RTSGAMEAGENT (Ours)	$0.93^{0.98}_{0.82}$	$0.93^{0.98}_{0.82}$	$0.90^{0.96}_{0.77}$	$1.00^{1.00}_{0.92}$	$0.77^{0.87}_{0.62}$
GPT-5.6-Luna	$0.47^{0.61}_{0.33}$	$0.50^{0.64}_{0.36}$	$0.83^{0.92}_{0.70}$	$0.50^{0.64}_{0.36}$	$0.00^{0.08}_{0.00}$
Qwen3.8-27B	$0.60^{0.73}_{0.45}$	$0.63^{0.76}_{0.48}$	$0.83^{0.92}_{0.70}$	$0.20^{0.34}_{0.11}$	$0.20^{0.34}_{0.11}$
Qwen3.5-397B	$0.53^{0.67}_{0.39}$	$0.67^{0.79}_{0.52}$	$0.87^{0.94}_{0.73}$	$0.27^{0.41}_{0.16}$	$0.03^{0.14}_{0.01}$
Muse-Spark-1.2	$1.00^{1.00}_{0.92}$	$0.73^{0.84}_{0.59}$	$0.80^{0.89}_{0.66}$	$0.27^{0.41}_{0.16}$	$0.23^{0.38}_{0.13}$
GPT-5.6-Terra	$0.73^{0.84}_{0.59}$	$0.37^{0.52}_{0.24}$	$0.90^{0.96}_{0.77}$	$0.30^{0.45}_{0.18}$	$0.13^{0.27}_{0.06}$
GLM-5.3-Flash	$0.93^{0.98}_{0.82}$	$0.83^{0.92}_{0.70}$	$1.00^{1.00}_{0.92}$	$0.53^{0.67}_{0.39}$	$0.27^{0.41}_{0.16}$
Grok-4.6	$1.00^{1.00}_{0.92}$	$0.80^{0.89}_{0.66}$	$1.00^{1.00}_{0.92}$	$0.47^{0.61}_{0.33}$	$0.23^{0.38}_{0.13}$
Gemini-3.7-Flash	$0.93^{0.98}_{0.82}$	$0.93^{0.98}_{0.82}$	$0.90^{0.96}_{0.77}$	$1.00^{1.00}_{0.92}$	$0.77^{0.87}_{0.62}$
Lv.2					
CoS (Ma et al., 2024)	$0.10^{0.23}_{0.04}$	—	—	—	—
LLM-PySC2 (Li et al., 2025)	$0.43^{0.58}_{0.30}$	$0.10^{0.23}_{0.04}$	$0.10^{0.23}_{0.04}$	$0.30^{0.45}_{0.18}$	$0.00^{0.08}_{0.00}$
SwarmBrain (Shao et al., 2024)	$0.40^{0.55}_{0.27}$	$0.00^{0.08}_{0.00}$	$0.33^{0.48}_{0.21}$	$0.43^{0.58}_{0.30}$	$0.00^{0.08}_{0.00}$
RTSGAMEAGENT (Ours)	$0.67^{0.79}_{0.52}$	$0.60^{0.73}_{0.45}$	$0.47^{0.61}_{0.33}$	$0.40^{0.55}_{0.27}$	$0.43^{0.58}_{0.30}$
GPT-5.6-Luna	$0.07^{0.18}_{0.02}$	$0.10^{0.23}_{0.04}$	$0.40^{0.55}_{0.27}$	$0.27^{0.41}_{0.16}$	$0.00^{0.08}_{0.00}$
Qwen3.8-27B	$0.00^{0.08}_{0.00}$	$0.13^{0.27}_{0.06}$	$0.03^{0.14}_{0.01}$	$0.17^{0.30}_{0.08}$	$0.03^{0.14}_{0.01}$
Qwen3.5-397B	$0.17^{0.30}_{0.08}$	$0.37^{0.52}_{0.24}$	$0.20^{0.34}_{0.11}$	$0.13^{0.27}_{0.06}$	$0.00^{0.08}_{0.00}$
Muse-Spark-1.2	$0.13^{0.27}_{0.06}$	$0.60^{0.73}_{0.45}$	$0.20^{0.34}_{0.11}$	$0.17^{0.30}_{0.08}$	$0.10^{0.23}_{0.04}$
GPT-5.6-Terra	$0.20^{0.34}_{0.11}$	$0.23^{0.38}_{0.13}$	$0.37^{0.52}_{0.24}$	$0.20^{0.34}_{0.11}$	$0.00^{0.08}_{0.00}$
GLM-5.3-Flash	$0.77^{0.87}_{0.62}$	$0.47^{0.61}_{0.33}$	$0.37^{0.52}_{0.24}$	$0.30^{0.45}_{0.18}$	$0.00^{0.08}_{0.00}$
Grok-4.6	$0.83^{0.92}_{0.70}$	$0.13^{0.27}_{0.06}$	$0.57^{0.70}_{0.42}$	$0.23^{0.38}_{0.13}$	$0.10^{0.23}_{0.04}$
Gemini-3.7-Flash	$0.67^{0.79}_{0.52}$	$0.60^{0.73}_{0.45}$	$0.47^{0.61}_{0.33}$	$0.40^{0.55}_{0.27}$	$0.43^{0.58}_{0.30}$
Lv.3					
CoS (Ma et al., 2024)	$0.00^{0.08}_{0.00}$	—	—	—	—
LLM-PySC2 (Li et al., 2025)	$0.03^{0.14}_{0.01}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$	$0.10^{0.23}_{0.04}$	$0.00^{0.08}_{0.00}$
SwarmBrain (Shao et al., 2024)	$0.07^{0.18}_{0.02}$	$0.00^{0.08}_{0.00}$	$0.30^{0.45}_{0.18}$	$0.13^{0.27}_{0.06}$	$0.00^{0.08}_{0.00}$
RTSGAMEAGENT (Ours)	$0.20^{0.34}_{0.11}$	$0.37^{0.52}_{0.24}$	$0.20^{0.34}_{0.11}$	$0.23^{0.38}_{0.13}$	$0.13^{0.27}_{0.06}$
GPT-5.6-Luna	$0.00^{0.08}_{0.00}$	$0.03^{0.14}_{0.01}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Qwen3.8-27B	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$	$0.10^{0.23}_{0.04}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Qwen3.5-397B	$0.00^{0.08}_{0.00}$	$0.13^{0.27}_{0.06}$	$0.10^{0.23}_{0.04}$	$0.07^{0.18}_{0.02}$	$0.00^{0.08}_{0.00}$
Muse-Spark-1.2	$0.07^{0.18}_{0.02}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$	$0.10^{0.23}_{0.04}$	$0.00^{0.08}_{0.00}$
GPT-5.6-Terra	$0.00^{0.08}_{0.00}$	$0.17^{0.30}_{0.08}$	$0.33^{0.48}_{0.21}$	$0.07^{0.18}_{0.02}$	$0.00^{0.08}_{0.00}$
GLM-5.3-Flash	$0.07^{0.18}_{0.02}$	$0.00^{0.08}_{0.00}$	$0.40^{0.55}_{0.27}$	$0.00^{0.08}_{0.00}$	$0.00^{0.08}_{0.00}$
Grok-4.6	$0.23^{0.38}_{0.13}$	$0.00^{0.08}_{0.00}$	$0.30^{0.45}_{0.18}$	$0.07^{0.18}_{0.02}$	$0.00^{0.08}_{0.00}$
Gemini-3.7-Flash	$0.20^{0.34}_{0.11}$	$0.37^{0.52}_{0.24}$	$0.20^{0.34}_{0.11}$	$0.23^{0.38}_{0.13}$	$0.13^{0.27}_{0.06}$

E.6 Anticipatory Wave Defense (AWD) — Adversarial Planning

Strategic demand. Plans must anticipate an opponent that acts concurrently, so preparations have to be made before threats materialize and are often costly in the short term. In BAR, counters to a specific threat, such as anti-air units or defensive structures, must be built in advance to be available when needed.

Objective. The agent must withstand successive enemy waves converging on its commander from different directions for 30 seconds. A run counts as a win if the commander survives all waves. The available action categories are **Build**, **Prod.**, and **Action**, and FoW is **disabled**.

Difficulty levels. The levels increase both the number of attack directions and the strength of the waves. Lv.1 sends waves from two directions, Lv.2 from three with the total enemy power increased by 50% over Lv.1, and Lv.3 from four with an increase of 100%. More directions leave fewer forces available per front, so the agent must spread its defenses rather than concentrate them.

Design rationale. The composition of each wave is given to the agent in advance, so the central question is whether it uses this information proactively rather than reacting to each wave as it arrives. Different compositions call for different unit types, defensive structures, and force placements, and preparation that has no immediate value, such as anti-air units before an air-heavy wave, pays off only if the agent plans ahead.

Game description. At the beginning of each episode, the agent receives a natural-language description of the survival objective and the compositions of the upcoming waves:

> *Withstand the enemy waves converging on your commander from different directions for 30 seconds. The composition of each wave is known in advance.*

E.7 Results with Confidence Intervals

Since each mini-game is evaluated over 30 runs per difficulty level, the reported win rates carry substantial sampling uncertainty. Tab. 13 therefore repeats the mini-game results of the main paper together with 90% Wilson score confidence intervals. The intervals are wide enough that many of the differences between models at a single level are not statistically distinguishable, and we accordingly base our claims on consistent trends across levels and mini-games rather than on individual cells.

F Detailed Procedure of RTSGameAgent

This section details the action space (Sec. F.1), trajectory planning and replanning (Sec. F.2), and inference loop (Sec. F.3) of RTSGameAgent introduced in Sec. 4.

F.1 Action Space

RTSGameAgent keeps the building construction and unit production actions of the benchmark interface (Eq. 1) and replaces per-unit *unit action* for combat forces with two group-level action types, *group assignment* and *group command*, so that the action space at interval t is

$$\mathcal{A}_t = \mathcal{A}_t^{\text{build}} \cup \mathcal{A}_t^{\text{produce}} \cup \mathcal{A}_t^{\text{assign}} \cup \mathcal{A}_t^{\text{command}}. \tag{5}$$

Building construction and unit production. Let $\mathcal{U}_t^{\text{ctor}}$ denote the available constructor units, $\mathcal{F}_t$ the active production buildings, and $\mathcal{O}(e)$ the build options of entity e in the encyclopedia of $\mathcal{K}$, *i.e.*, the building types a constructor can build or the unit types a production building can produce. With $\mathcal{X} \times \mathcal{Y}$ denoting the coordinate grid spanning $(0, 0)$ to $(100, 100)$,

$$\mathcal{A}_t^{\text{build}} = \big\{(u, b, x, y) \;\big|\; u \in \mathcal{U}_t^{\text{ctor}},\ b \in \mathcal{O}(u),\ (x, y) \in \mathcal{X} \times \mathcal{Y}\big\}, \tag{6}$$

$$\mathcal{A}_t^{\text{produce}} = \big\{(f, p) \;\big|\; f \in \mathcal{F}_t,\ p \in \mathcal{O}(f)\big\}. \tag{7}$$

Both action types are inherited unchanged from the base interface, and their low-level execution is handled by the game engine.

Group assignment. Let $\mathcal{U}_t^{\text{combat}}$ denote the active combat units and $\mathcal{G}_t$ the group identifiers available at interval t, including newly created ones. A group-assignment action associates a subset of combat units with a group:

$$\mathcal{A}_t^{\text{assign}} = \mathcal{G}_t \times 2^{\mathcal{U}_t^{\text{combat}}}, \tag{8}$$

where each unit belongs to at most one group, so that assigning a unit removes it from its previous group. Unlike predefined combat groups (Li et al., 2025), groups are created, updated, or dissolved by the VLM according to the current strategy and game state. The textual observation $\mathcal{W}(s_t)$ reports the current group membership, unit counts, and unassigned combat units, so that subsequent assignments can build on the existing grouping.

Group command. For each group $g \in \mathcal{G}_t$, a group command specifies a mode from $\mathcal{C} = \{\texttt{move}, \texttt{move_force}, \texttt{stop}\}$ and, when needed, a target coordinate:

$$\mathcal{A}_t^{\text{command}} = \mathcal{G}_t \times \mathcal{C} \times \big((\mathcal{X} \times \mathcal{Y}) \cup \{\varnothing\}\big), \tag{9}$$

where $\varnothing$ denotes no target, as for `stop`. Under `move`, a group travels toward its target, and under `stop`, it holds its current location. In both modes, engagement is handled by the execution controller, which switches a group to a `fight` state upon enemy contact and resumes its command once the engagement ends. Under `move_force`, a group instead pushes through toward its target without engaging, which is useful for repositioning across contested terrain. Since `fight` is never issued by the VLM, it is excluded from $\mathcal{C}$. Consequently, the number of commands the VLM issues scales with the number of groups $|\mathcal{G}_t|$ rather than the number of combat units, while the VLM retains control over group membership, command modes, targets, and conditional trajectories.

F.2 Trajectory Planning and Replanning

Command trajectories. Since the VLM is invoked only once per interval, a single command would leave a group unable to react until the next call. Instead, the VLM outputs for each group g a condition-dependent command trajectory, represented as a directed acyclic graph (DAG) $\mathcal{T}_t^g = (\mathcal{V}_t^g, \mathcal{E}_t^g)$ whose nodes hold group commands and whose edges are labeled with transition conditions. Execution starts at the root node. Whenever the condition on an outgoing edge of the active node is satisfied, the controller activates the successor node without another VLM call; otherwise, the active command remains in effect.

Transition conditions. Rather than free-form text, each condition is one of four predefined primitives whose parameter is set by the VLM, allowing the controller to evaluate conditions unambiguously:

- `idle_for`: all units in the group have no remaining orders, typically after completing the current command.
- `hp_below`: the average health of units in the group falls below a threshold.
- `group_size_below`: the number of units in the group falls below a threshold.
- `timeout`: a given amount of time has elapsed since the current node became active.

For example, a group can advance to a rally point and, once `idle_for` holds, proceed to an enemy base, while an `hp_below` edge redirects it to the home base if it sustains heavy damage.

Replanning triggers. Trajectories cover changes that can be anticipated at decision time. For sudden changes that require a new strategic decision, three triggers with fixed thresholds are monitored over the entire game state:

- `enemy_surge`: the number of visible enemy units increases by at least 50% and at least five units since the most recent inference.
- `army_loss`: the number of allied combat units decreases by at least 30% and at least five units since the most recent inference.
- `commander_hp`: the commander's health falls below 50% of its maximum.

Algorithm 1 Inference loop of RTSGAMEAGENT.

Require: static game knowledge $\mathcal{K}$, VLM policy π, call interval Δ
Notation: $\mathcal{T}_t^g$ is the trajectory of group g at call t, a DAG whose nodes are commands $c \in \{\texttt{move}, \texttt{move_force}, \texttt{stop}\}$ and whose edges carry transition conditions.

$\mathcal{L}_0 \leftarrow \varnothing$; $\mathcal{S}_1 \leftarrow \varnothing$
for $t = 1, 2, \ldots$ until the game ends **do**
  Pause the game and read the engine state s_t
  $o_t \leftarrow \big(v_t, \mathcal{W}(s_t)\big)$ ▷ Observation, restricted to allied line of sight under FoW
  $(m_t, \mathcal{L}_t) \leftarrow \mathrm{VLM}(\mathcal{S}_t, \mathcal{L}_{t-1})$ ▷ Memory phase, text-only
  $\mathcal{S}_{t+1} \leftarrow \varnothing$
  $a_t \leftarrow \pi(o_t, m_t \mid \mathcal{K})$ ▷ Decision phase
  Dispatch building and production orders to the game engine
  Update groups and trajectories $\{\mathcal{T}_t^g\}$; activate each root node
  Resume the game
  repeat ▷ Trajectory execution without further VLM calls
    Advance the game; append triggered event logs to $\mathcal{S}_{t+1}$
    for each group g **do**
      if a condition on an outgoing edge of the active node of $\mathcal{T}_t^g$ holds **then**
        Activate the corresponding successor node
      end if
    end for
  until Δ has elapsed **or** a critical trigger fires ▷ The latter initiates replanning
end for

Combining relative and absolute thresholds keeps the first two triggers insensitive to small fluctuations regardless of army size. When any trigger fires, the game is paused and a new inference cycle begins immediately, replacing the current group assignments and trajectories.

F.3 INFERENCE LOOP

Algorithm 1 summarizes the inference loop. Following the benchmark interface, the game advances for a fixed interval Δ between scheduled inferences and is paused during inference. Before the game begins, the agent receives the static game knowledge $\mathcal{K}$, consisting of the scenario description, team configuration, and unit and building encyclopedia, which remains fixed throughout the episode. The long-term memory, short-term memory, and set of groups all start empty.

Observation. At each interval t, the agent constructs $o_t = \big(v_t, \mathcal{W}(s_t)\big)$ from the engine state s_t. The visual observation v_t consists of a global minimap and up to four local camera views, selected from candidate locations in the following fixed order of priority: (i) recent combat locations, (ii) the commander's location when its health is below 50%, the same threshold as the `commander_hp` trigger, (iii) visible enemy force clusters, (iv) allied group clusters, (v) the commander's location when its health is 50% or above, and (vi) the allied home base. Spatially close candidates are merged, and up to four of the highest-priority locations are selected. With FoW enabled, both the minimap and the local views are restricted to allied line of sight. The textual observation $\mathcal{W}(s_t)$, extracted by the Python wrapper, summarizes resources, buildings and production states, allied units and groups, visible enemies, and pending actions.

Memory phase. Between VLM calls, event-triggered logs accumulate in the short-term memory $\mathcal{S}_t$. A log entry is recorded when (i) enemy units are observed, (ii) allied units, including the commander, take damage or are destroyed, (iii) a new unit is completed, (iv) a unit receives a new order, such as moving to a target coordinate, (v) a resource reaches its storage cap, indicating a surplus, or (vi) a resource is depleted, indicating a shortage. At each inference cycle, the policy VLM, prompted with text only, consolidates $\mathcal{S}_t$ with the previous long-term memory $\mathcal{L}_{t-1}$, producing the updated memory $\mathcal{L}_t$ and the decision-relevant context m_t:

$$(m_t, \mathcal{L}_t) = \mathrm{VLM}\big(\mathcal{S}_t, \mathcal{L}_{t-1}\big). \tag{10}$$

$\mathcal{S}_t$ is then cleared, so that transient events do not accumulate indefinitely, while the persistent summaries in $\mathcal{L}_t$ retain strategically relevant information throughout the game.

Table 14: Structural summary of the adapted baselines. SwarmBrain calls the Overmind and the SC2 Brain; LLM-PySC2 calls the Commander, the Developer, and one CombatGroup per squad, where N is the number of combat squads.

	CoS (Ma et al., 2024)	SwarmBrain (Shao et al., 2024)	LLM-PySC2 (Li et al., 2025)
LLM calls per step	1	2	N+2
Memory	last 4 frames	last 5 directives	none
Unit grouping	none	collective, label-addressed	per unit archetype
Unit-level control	shared rules	ReflexNet + shared rules	shared rules
Army engagement	scripted push	SC2 Brain	CombatGroups

Decision phase. The VLM policy then produces the action plan $a_t = \pi(o_t, m_t \mid \mathcal{K})$, comprising building and production orders, group assignments, and a command trajectory for each group (Sec. F.1 and Sec. F.2). Building and production orders are executed through the base interface, while group assignments and trajectories are passed to the execution controller, which carries them out without further model calls until Δ elapses or a critical trigger fires.

F.4 Adapting StarCraft II LLM Agents to BAR

CoS (Ma et al., 2024), SwarmBrain (Shao et al., 2024), and LLM-PySC2 (Li et al., 2025) were designed for StarCraft II (SC2) and depend on its interfaces, race-specific mechanics, and worker-harvesting economy, so running them on BAR requires re-implementation. This section describes each re-implementation component by component, showing that it preserves the mechanisms its original paper identifies as its contribution and stating every deviation from the original explicitly.

Shared protocol. All agents, including ours, share the observation interface, action executor, unit-level control rules, backbone LLM, decoding configuration, and decision interval of the main experiments. On this shared substrate, each baseline follows its original paper in prompt structure, LLM roles, memory, grouping, and reflex rules, except for the deviations noted below (Table 14). No baseline uses the memory or replanning modules of our agent, and race-specific prompt content is rewritten for BAR while the prompt structure is kept.

CoS (Ma et al., 2024)**.** In TextStarCraft II, the LLM selects only macro actions while scripts handle all unit control. Each observation is summarized, K summaries are aggregated, and the LLM reasons through six phases, from situation overview to decision-making, before extracting actions into a queue. Our port keeps these elements. The structured text observation serves as the single-frame summary, the LLM receives the current frame with the preceding $K{=}4$ frames, and the prompt enforces the six phases in order. Its action space is restricted to position-free construction and production, as research has no BAR counterpart, where technology advances through higher-tier factories. The scripted engagement becomes a fixed-timing push: at 20 in-game minutes, all combat units attack a target sampled from the enemy Commanders' spawn positions, re-issued every 5 minutes so that new units join. *Deviation:* Instead of querying every K steps and executing a queue of K actions, we query at every decision step over a sliding window of K frames, keeping CoS on the shared decision interval.

SwarmBrain (Shao et al., 2024)**.** SwarmBrain pairs an LLM-based Overmind Intelligence Matrix with the Swarm ReflexNet, rule-based state machines that act without waiting for the LLM. Within the Matrix, the Overmind Brain formulates a strategy, a text memory keeps it consistent, the SC2 Brain translates it into commands, and a Command Center validates them, with locations referenced by mineral-field labels. Our port keeps this structure. Each cycle runs the Overmind and then the SC2 Brain on its directive, giving two calls per step, and the five most recent directives serve as the strategy memory. Metal spots, the BAR counterpart of mineral fields, are labeled at every frame and referenced by both roles, and our JSON validator replaces the Command Center. Since BAR has neither worker harvesting nor supply units, we port only the ReflexNet's flee transition, retreating a damaged combat unit near an enemy. *Deviation:* The original SC2 Brain is text-only and forms collectives by unit type and location; ours receives the shared visual observation and commands the groups of our interface.

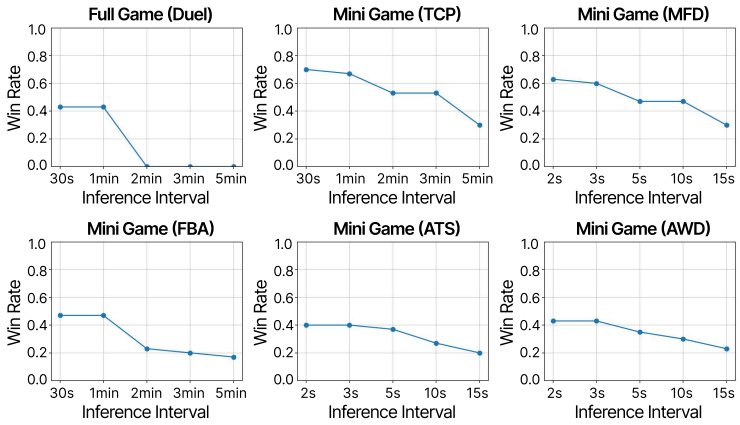


Figure 12: **Sensitivity to the inference interval.** Win rate of RTSGAMEAGENT with Gemini-3.7-Flash in six Lv.2 scenarios under different inference intervals, *i.e.*, the game time between VLM calls (30 games per point). Shorter intervals allow more frequent decisions but require more VLM calls; win rate saturates at the two shortest intervals and declines beyond them. We use 1 min for full games, TCP, and FBA, and 3 s for MFD, ATS, and AWD.

LLM-PySC2 (Li et al., 2025)**.** LLM-PySC2 defines a Commander for combat deployment, a Developer for economy, CombatGroups for micro-operations, and a Builder for building placement, all queried in independent threads. The Commander directs the CombatGroups through messages delivered at the next step. Our port keeps these elements. The Commander only sends messages, the Developer handles construction and production, and one CombatGroup per unit-archetype group moves its own group, giving $N+2$ concurrent calls per step. Folding the Builder into the Developer corresponds to Easy Build mode, so our setting matches the paper's Standard Control + Easy Build configuration. Each role observes what it needs, namely group centroids and alerts for the Commander, base state and army totals for the Developer, and nearby units with a screen crop for each CombatGroup, and all roles receive the minimap. *Deviation:* The original CombatGroups act over the full PySC2 action space, including unit skills; ours issue group-level movement, and unit-level engagement follows the shared rules.

F.5 INFERENCE INTERVAL ANALYSIS

The inference interval, *i.e.*, the game time between consecutive VLM calls, trades off decision frequency against inference cost. Since the game is paused during inference, a shorter interval does not penalize slow responses but lets the agent revise its decisions more often, at the cost of more VLM calls. We therefore set the interval according to the temporal demands of each setting, sharing one interval within each group below.

Full games (1 min). Full games span long horizons dominated by economic expansion and strategic planning. Sudden events within an interval are handled by command trajectories and replanning triggers (Sec. F.2), so frequent VLM calls are unnecessary.

Planning-oriented mini-games (1 min). TCP requires meeting production and economy targets, and FBA requires destroying a fortified enemy base. Both depend on long-term planning, such as sequencing builds or preparing an assault, rather than on moment-to-moment reactions.

Defense-oriented mini-games (3 s). MFD, ATS, and AWD require repelling incoming attacks, during which the situation changes rapidly and forces must be redeployed frequently.

Sensitivity results. In Fig. 12, win rate in every setting saturates at the two shortest intervals and declines as the interval grows further, most sharply in full games. The chosen intervals all lie within this saturated range and thus retain near-best performance, whereas longer intervals degrade it.

G Details about Evaluation Metrics for RTSGameBench

We use win rate (WR) as the primary metric for agent evaluation, together with six auxiliary metrics for a detailed analysis of agent behavior in full games.

Win rate (WR). WR is the primary metric for all full-game match-ups and mini-games, defined as the fraction of evaluation runs in which the agent satisfies the win condition:

$$\mathrm{WR} = \frac{\#\,\text{wins}}{\#\,\text{runs}}. \tag{11}$$

In team match-ups, a run counts as a win when the agent's team wins. The win condition of each mini-game is described in Sec. E.

Motivation for auxiliary metrics. WR reflects only the final outcome and barely distinguishes agents when it is close to zero or one. To analyze *how* agents play in full games, we measure four dimensions of gameplay: *resources* (resources gathered, RG; resources spent, RS), *army* (army strength, AS; unit quality, UQ), *battle* (enemies destroyed, ED), and *visibility* (vision coverage, VC). Raw in-game statistics, however, are hard to compare across runs. Their scale depends on the match-up, and since they accumulate over time, they conflate performance with how long a player stays in the game, so that players eliminated early would appear weak in every dimension and inflate the others' shares. We therefore express each quantity as a rate over each player's alive span and report the agent's normalized share among all players, leaving survival itself to WR. Per-unit averages, which are already independent of match length, are used without this conversion.

Share normalization. Let $T_i = [t_0, t_i^{\text{end}}]$ with $t_i^{\text{end}} = \min(t_{\text{final}}, t_i^{\text{death}})$ denote the alive span of player i, where t_0 and t_{final} are the start and end of the match, and let x_i^{M} denote its per-player quantity for metric M, defined below. The agent's share among all N players, counting each ally and opponent separately, is

$$p^{\mathrm{M}} = \frac{x_1^{\mathrm{M}}}{\sum_{i=1}^{N} x_i^{\mathrm{M}}}, \tag{12}$$

with player 1 denoting the agent. Since the even share $1/N$ varies across match-ups, we report the odds-normalized share

$$\mathrm{M} = f\big(p^{\mathrm{M}}; N\big), \qquad f(p; N) = \frac{\rho}{1+\rho}, \quad \rho = \frac{(N-1)\,p}{1-p}, \tag{13}$$

which maps $p = 0, 1/N$, and 1 to 0, 0.5, and 1, so that 0.5 denotes an even share in every match-up. Equivalently, $\mathrm{M} = x_1^{\mathrm{M}}/(x_1^{\mathrm{M}} + \bar{x}_{-1}^{\mathrm{M}})$, where $\bar{x}_{-1}^{\mathrm{M}}$ is the mean over the other $N-1$ players, so each metric compares the agent with an average other player. For $N = 2$, f reduces to the identity, *i.e.*, a share between two sides is already anchored at 0.5 and is reported as is.

Per-player quantities. All quantities are expressed in *power*, BAR's internal value scale, which converts energy into metal-equivalent units at a ratio of 60 to 1. The power of an entity u is thus $\beta(u) = c_{\text{metal}}(u) + c_{\text{energy}}(u)/60$, where c_{metal} and c_{energy} denote its metal and energy costs, and the same conversion is applied to resource income and spending. For player i, let $\phi_i(t)$ denote its power income rate, $S_i(t)$ its cumulative power spent, $\mathcal{U}_i^{\text{combat}}(t)$ its surviving combat units at time t, $\mathcal{P}_i$ the combat units it produced within T_i, and $\mathcal{D}_i$ the enemy units and buildings it destroyed within T_i according to kill attribution. Quantities that accumulate over time are divided by $|T_i|$ to obtain rates over the alive span:

$$x_i^{\mathrm{RG}} = \frac{1}{|T_i|}\int_{T_i} \phi_i(t)\,dt, \qquad x_i^{\mathrm{RS}} = \frac{S_i(t_i^{\text{end}})}{|T_i|}, \tag{14}$$

$$x_i^{\mathrm{AS}} = \frac{1}{|T_i|}\int_{T_i} \sum_{u \in \mathcal{U}_i^{\text{combat}}(t)} \beta(u)\,dt, \qquad x_i^{\mathrm{ED}} = \frac{1}{|T_i|}\sum_{u \in \mathcal{D}_i} \beta(u), \tag{15}$$

$$x_i^{\mathrm{UQ}} = \frac{1}{|\mathcal{P}_i|}\sum_{u \in \mathcal{P}_i} \beta(u). \tag{16}$$

AS measures the average power of the combat force a player maintains, whereas UQ measures the average power per produced unit and thus reflects how advanced its unit composition is. We

measure UQ with power rather than BAR's discrete tech tiers (T1–T3), because units within the same tier differ considerably in cost and capability. Tiers register only transitions between tiers and miss shifts toward stronger units within a tier, whereas power places every unit on a continuous scale that still increases with tier and is shared with the other metrics. AS and UQ count only combat units, as the army dimension concerns the force the agent can deploy, whereas investment in economic and defensive structures is reflected in RS.

Vision coverage. Since allies share vision, VC is computed as a share among alliances rather than players. Let K denote the number of alliances in a match, *i.e.*, $K = 2$ in Duel and team matchups and $K = 4$ in Free-for-All, and let $\bar{\eta}_k$ denote the fraction of the map within the vision of alliance k, averaged over the part of the agent's alive span T_1 during which alliance k has surviving members, with $k = 1$ denoting the agent's alliance. Restricting each average to the period in which the alliance survives prevents alliances eliminated earlier in Free-for-All from inflating the agent's share, consistent with the alive-span normalization of the other metrics. VC is defined as

$$\mathrm{VC} = f\left(\frac{\bar{\eta}_1}{\sum_{k=1}^{K} \bar{\eta}_k};\, K\right), \tag{17}$$

which reduces to the alliance share itself when $K = 2$.

H Details of the Self-Evolving Game Generation Framework

This section details the self-evolving game generation framework introduced in Sec. 3.3; its full procedure (Sec. H.1), the evaluation of generated games (Sec. H.2), an ablation of its self-evolution mechanisms (Sec. H.3), representative generation examples (Sec. H.4), and agent evaluation on the generated games (Sec. H.5).

H.1 Algorithm Details

Algorithm 2 presents the full procedure of the framework, and we detail each component below.

Inputs and initialization. All four agents, *i.e.*, the PM, designer, developer, and analyst, are driven by Gemini-3.8-Flash. The framework takes as input a user query q and a shared knowledge database $\mathcal{D}$, which holds validated GDDs and rule sets, the validation rubrics $\mathcal{R}$, the game assets available in BAR, and the five human-designed diagnostic mini-games (Sec. 3.2) at their three difficulty levels, which serve as reference scenarios $\mathcal{S}_{\text{ref}}$ for difficulty scaling. GDDs, rule sets, and rubrics start empty and accumulate only through self-evolution, whereas the game assets and $\mathcal{S}_{\text{ref}}$ remain fixed. Because few validation criteria apply across stages, the rubrics are maintained separately for each stage, $\mathcal{R} = \{\mathcal{R}^{(2)}, \mathcal{R}^{(3)}, \mathcal{R}^{(4)}\}$, each organized as a two-level hierarchy that groups individual rubrics under broad categories. Before each validation, the PM inspects both the categories and their rubrics for that stage and selects those relevant to the current game (Fig. 21), denoted $\mathcal{R}_g \subseteq \mathcal{R}^{(2)}$, $\mathcal{R}_r \subseteq \mathcal{R}^{(3)}$, and $\mathcal{R}_s \subseteq \mathcal{R}^{(4)}$.

An artifact buffer $\mathcal{A}$ and a feedback log $\mathcal{F}$ start empty; $\mathcal{A}$ accumulates validated outputs for downstream reuse, and $\mathcal{F}$ records all analyst feedback, including failed attempts, for rollbacks and the final retrospective analysis.

Rollbacks and aborts. Stages 2–4 share the same control flow for validation failures. When a stage fails, the PM invokes MetaFeedback, which reviews $\mathcal{F}$, including the failure history and the approaches attempted so far, and rolls the pipeline back to the stage it identifies as the source of the failure, either the failing stage or an earlier one, with corrective guidance $\mathrm{fb}^\star$ for the next attempt. Artifacts of the stages being redone are discarded from $\mathcal{A}$. The PM can perform at most three rollbacks per generation; if validation still fails, generation is aborted and outputs what it has produced so far, *i.e.*, the latest game script if Stage 4 has been reached and the earlier artifacts otherwise. Such *aborted* generations are included in the evaluation (Sec. H.2), and experience-driven evolution is expected to reduce their number over successive cycles.

Stage 1: scenario brief generation. The designer engages in multi-turn dialogue with the user to clarify the intent of q, producing a structured scenario brief b that specifies the map, game composition, enemy behavior rules, win/loss conditions, and proposed ratings of the five strategic demands (Sec. 3.2) on a 0–5 scale. The PM then compares b with all GDDs in $\mathcal{D}$ and judges whether

Algorithm 2 Self-Evolving Game Generation Framework

Require: User query q; shared knowledge database $\mathcal{D}$ of GDDs, rule sets, and stage-wise rubrics $\mathcal{R}$, all initially empty, and fixed reference scenarios $\mathcal{S}_{\text{ref}}$; rollback budget $B{=}3$
Ensure: Mini-games $\mathcal{G}_{1:3}$ at Lv.1–3 with status `ok`, or, if aborted, the outputs produced so far with status `aborted`; updated $\mathcal{D}$ and $\mathcal{R}$

$\mathcal{A} \leftarrow \emptyset$, $\mathcal{F} \leftarrow \emptyset$, $n \leftarrow 0$, $\text{fb}^\star \leftarrow \emptyset$ ▷ Artifacts, feedback log, rollback count, guidance
// Stage 1: Scenario Brief Generation
$b \leftarrow$ DESIGNER.CLARIFY(q) ▷ Brief with map, composition, and demand ratings
$k \leftarrow 2$ ▷ Current stage
if PM.MATCH$(b, \mathcal{D})$ finds a reusable GDD **then** ▷ Compare b with all stored GDDs
  Retrieve $\{\text{GDD}, \mathcal{C}, \text{Rules}\}$ from $\mathcal{D}$ into $\mathcal{A}$; $k \leftarrow 4$ ▷ Fast-track past Stages 2–3
end if
// Stages 2–4 with a Global Rollback Budget
while $k \leq 4$ **do**
  if $k = 2$ **then** ▷ GDD generation
    $(\text{GDD}, \mathcal{C}) \leftarrow$ DESIGNER.EXPAND$(b, \text{fb}^\star)$ ▷ Finalize demand ratings
    $\mathcal{R}_g \leftarrow$ PM.SELECTRUBRICS$(\mathcal{R}^{(2)}, b)$
    $(\text{pass}, \text{fb}) \leftarrow$ ANALYST.VALIDATE$(\text{GDD}, \mathcal{R}_g)$
  else if $k = 3$ **then** ▷ Rule set construction
    $\text{Rules}_{\text{reuse}} \leftarrow$ PM.RETRIEVERULES$(\text{GDD}, \mathcal{D})$ ▷ Verified implementations, if any
    $(\text{Rules}, \mathcal{T}_r) \leftarrow$ DEVELOPER.IMPLEMENT$(\text{GDD}, \text{Rules}_{\text{reuse}}, \text{fb}^\star)$ ▷ New rules only
    $\mathcal{R}_r \leftarrow$ PM.SELECTRUBRICS$(\mathcal{R}^{(3)}, \text{GDD})$
    $(\text{pass}, \text{fb}) \leftarrow$ ANALYST.VALIDATE$(\text{Rules}, \mathcal{R}_r, \mathcal{T}_r)$ ▷ Rule-based tests in engine
  else ▷ Game implementation and verification
    $(\mathcal{G}, \mathcal{T}_s) \leftarrow$ DEVELOPER.CONFIGURE$(b, \mathcal{A}, \text{Assets}, \text{fb}^\star)$ ▷ Medium difficulty, map in b
    $\mathcal{R}_s \leftarrow$ PM.SELECTRUBRICS$(\mathcal{R}^{(4)}, \mathcal{A})$
    $(\text{pass}, \text{fb}) \leftarrow$ ANALYST.VALIDATE$(\mathcal{G}, \mathcal{R}_s \cup \mathcal{C}, \mathcal{T}_s)$ ▷ Rubrics, checklist, and tests
  end if
  $\mathcal{F} \leftarrow \mathcal{F} \cup \{\text{fb}\}$
  if pass **then**
    Add the stage output to $\mathcal{A}$ if $k \leq 3$; $k \leftarrow k + 1$; $\text{fb}^\star \leftarrow \emptyset$
  else if $n < B$ **then**
    $(k, \text{fb}^\star) \leftarrow$ PM.METAFEEDBACK$(\mathcal{F})$ ▷ Roll back to the failing or an earlier stage
    $\mathcal{A} \leftarrow \mathcal{A}_{<k}$; $n \leftarrow n + 1$ ▷ Discard artifacts of stages $\geq k$
  else
    return (outputs produced so far, `aborted`) ▷ Budget exhausted
  end if
end while
// Difficulty Scaling (No Further Validation)
$\mathcal{G}_{1:3} \leftarrow$ DEVELOPER.SCALE$(\mathcal{G}, \mathcal{S}_{\text{ref}})$ ▷ Lv.1–3 at once; $\mathcal{G}$ as medium
// Self-Evolution Phase
$\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{A}$ ▷ Commit GDD, checklist, and rule set
$\mathcal{R} \leftarrow$ PM.RETROSPECTIVE$(\mathcal{R}, \mathcal{F})$ ▷ Update rubrics within each stage's hierarchy
return $(\mathcal{G}_{1:3}, \texttt{ok})$

any is similar enough to reuse, deciding case by case whether a partial match suffices. If so, the pipeline *fast-tracks* to Stage 4 by retrieving that GDD, its checklist, and its rule set from $\mathcal{D}$, bypassing Stages 2 and 3.

Stage 2: GDD generation. The designer expands b into a full game design document (GDD) that finalizes the demand ratings, with the focal demand rated highest, and specifies the rule components governing game behavior, *e.g.*, unit spawning conditions and termination criteria, together with a checklist $\mathcal{C}$ for validating the final game. The analyst validates the GDD against $\mathcal{R}_g$, returning a binary pass signal pass_g and structured feedback fb_g. Once the GDD passes, it is stored in $\mathcal{A}$ together with $\mathcal{C}$.

Stage 3: rule set construction. The PM first retrieves verified implementations of the rules required by the GDD from $\mathcal{D}$, when available. The developer then implements the remaining rules as Lua

scripts (Ierusalimschy, 2006) for in-engine execution and writes test code $\mathcal{T}_r$ for the rule set. These simulation tests run in the BAR engine without a playing agent: each sets up a predefined situation and checks, in a rule-based manner, whether the specified behavior occurs, *e.g.*, whether units spawn at the specified time. The analyst validates the rule set by running $\mathcal{T}_r$ and checking it against $\mathcal{R}_r$, returning pass_r and fb_r, and the passing rule set is appended to $\mathcal{A}$.

Stage 4: game implementation and verification. Based on b and the artifacts in $\mathcal{A}$, the developer retrieves game assets, *e.g.*, unit configurations, from $\mathcal{D}$ and assembles an executable game script $\mathcal{G}$ at medium difficulty on the map specified in b, setting unit placement, end conditions, and rule parameters. It also writes simulation tests $\mathcal{T}_s$ for the assembled game in the same form as in Stage 3. The analyst validates $\mathcal{G}$ by running $\mathcal{T}_s$ and checking it against $\mathcal{R}_s$ and the checklist $\mathcal{C}$, returning pass_s and fb_s.

Difficulty scaling. Once $\mathcal{G}$ passes validation, the developer generates the scripts at all three difficulty levels, $\mathcal{G}_{1:3}$ for Lv.1–3, at once, taking $\mathcal{G}$ as the medium-difficulty reference and using $\mathcal{S}_{\text{ref}}$ as in-context examples that pair each reference scenario's three levels with the rationale for their difficulty. Elements shared across levels, *i.e.*, the map, match format, active mechanics, and inference interval, are kept fixed, and only difficulty parameters, *e.g.*, the number of enemy waves, unit counts, resource caps, time limits, and enemy AI aggressiveness, are varied within the ranges observed in $\mathcal{S}_{\text{ref}}$. Scaling concludes generation without further validation, as the scaled scripts share the rule set validated for $\mathcal{G}$ and differ from it only in difficulty parameters. The resulting games are evaluated as described in Sec. H.2.

Self-evolution phase. Upon successful generation, two self-evolution mechanisms are triggered. First, the validated artifacts in $\mathcal{A}$, *i.e.*, the GDD, its checklist, and the rule set, are committed to the database, $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{A}$, so that future queries can reuse them to fast-track or partially skip stages. The final scripts at the three difficulty levels are not stored, so $\mathcal{S}_{\text{ref}}$ remains the only reference for difficulty scaling. Second, the PM conducts a *retrospective analysis* over $\mathcal{F}$, including failed attempts, identifying systematic discrepancies between verification outcomes and quality expectations and updating the rubrics accordingly, $\mathcal{R} \leftarrow \text{PM.RETROSPECTIVE}(\mathcal{R}, \mathcal{F})$. Because updates are applied within the relevant category of each stage's hierarchy, validation experience accumulates where it applies instead of in a single monolithic rubric. Together, these mechanisms improve both generation efficiency and output quality over successive cycles (Sec. 5.3), and each contributes to both gains (Sec. H.3).

H.2 Evaluation of Generated Games

Evaluating generated games is challenging for several reasons. First, there is no reference against which a generated game can be compared: each game is created on demand from a free-form query, so its correctness can only be judged against the intent expressed in that query. Second, whether a game realizes this intent becomes observable only in play. A game script can pass static inspection and even run without errors while failing to produce the intended dynamics, and many intended behaviors, such as victory and defeat, arise only in particular situations whose occurrence depends on how the game is played, so a single play-through may never reach them. Third, human judgment captures intent most faithfully, but it is costly to scale and does not by itself indicate which behaviors fail. Finally, the framework itself can fail: a generation may abort before producing a complete game, and one that succeeds may do so only after costly rollbacks. We therefore evaluate generated games along four complementary criteria: *playability*, a coarse check of whether generation yields a game that can be launched; *behavioral fidelity*, a fine-grained check of whether the behaviors expected from the query and GDD actually occur in play; *token usage*, the inference cost of producing a game; and *human preference*, a judgment of query alignment, fun and balance, and completeness by human raters. All four are measured for generation both with and without experience-driven evolution (Sec. 5.3) and reported per batch of five consecutive generation cycles, so that the effect of accumulated experience can be traced over successive cycles. Aborted generations are included in every criterion, so that failures are reflected in the results rather than excluded.

Algorithm 3 Behavioral Fidelity Evaluation

Require: User query q; game design document GDD; game scripts $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$ for the three difficulty levels, or the latest (possibly incomplete) $\mathcal{G}$ if aborted during Stage 4; test agent π (RTSGAMEAGENT, playing to win); rollout budget $M{=}6$

Ensure: Behavioral fidelity score $y \in [0, 1]$, or 0 if no game script exists

if no game script exists **then** ▷ Aborted before Stage 4
  return $y = 0$ ▷ No game to play
end if
// Checklist Construction
$\mathcal{E} \leftarrow$ BUILDCHECKLIST(q, GDD) ▷ Behaviors the game must exhibit
  $\mathcal{E}_{\text{init}}$: the game starts with the specified rosters and map/zones
  $\mathcal{E}_{\text{rule}}$: per required rule, it fires on time and takes its intended effect
  $\mathcal{E}_{\text{end}}$: victory (defeat) is declared when its condition holds
  $\mathcal{E}_{\text{pace}}$: up to three expected events on the pacing timeline occur
// Rollouts with Privileged Logging
$\Gamma \leftarrow \{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3\}$, or $\{\mathcal{G}\}$ if aborted
$R \leftarrow M/|\Gamma|$ ▷ 2 per level, or 6 if aborted
for all $\mathcal{G}_\ell \in \Gamma$ and $i = 1, \ldots, R$ **in parallel do**
  $\tau_{\ell,i} \leftarrow$ ROLLOUT$(\mathcal{G}_\ell, \pi)$ ▷ Full simulator state until the game ends or crashes
end for
// Checklist Matching
for all $e \in \mathcal{E}$ and rollouts $\tau_{\ell,i}$ **do**
  $v_{e,\ell,i} \leftarrow$ JUDGE$(e, \tau_{\ell,i}) \in \{0, 1\}$ ▷ 1 if the expected behavior is found
end for
$y \leftarrow \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \max_{\ell,i} v_{e,\ell,i}$ ▷ An item passes if found in any rollout
return y

H.2.1 PLAYABILITY

Playability is a coarse check of whether generation yields a game that can be launched at all. For each batch, it is computed as

$$\text{Playability} = \frac{\#\text{ games that launch successfully}}{\#\text{ generation attempts}}.$$

Each game is launched in the BAR engine once at each of the three difficulty levels, or once from its latest script if aborted, and counts as launching successfully if the game starts in every launch; what happens after the game starts is not considered. The denominator includes every generation attempt, including aborted ones: a generation aborted before Stage 4 produces no game and counts as a failure, whereas an aborted game whose latest script still launches counts as playable. Playability does not check whether the game behaves as intended; a game may launch even if its enemy waves never spawn or a rule script fails midway. Such failures are captured by behavioral fidelity.

H.2.2 BEHAVIORAL FIDELITY

Behavioral fidelity starts from the premise that a user's intent is ultimately realized through the in-game behaviors of the generated game. A game aligned with its query exhibits the behaviors that the query and its GDD call for, and it can do so only if its implementation functions correctly. We check whether these expected behaviors actually occur when the game is played and summarize the result as a score from 0 to 1, *i.e.*, the checklist pass rate, which reflects both alignment with the user's intent and functionality (Fig. 13; Algorithm 3). Aborted generations are scored in the same way whenever their latest script launches, as described below; only generations that yield no launchable game receive a score of 0. We report the mean and standard deviation of the scores over the five generations in each batch, including aborted ones.

Checklist construction. Gemini-3.8-Flash reads the user query and the GDD and constructs an evaluation checklist $\mathcal{E}$, typically of around ten behaviors that the game must exhibit in play, separately from the checklist used for validation during generation. Because it is derived from the GDD, the same checklist applies to all three difficulty levels. It covers the following categories: *rosters*

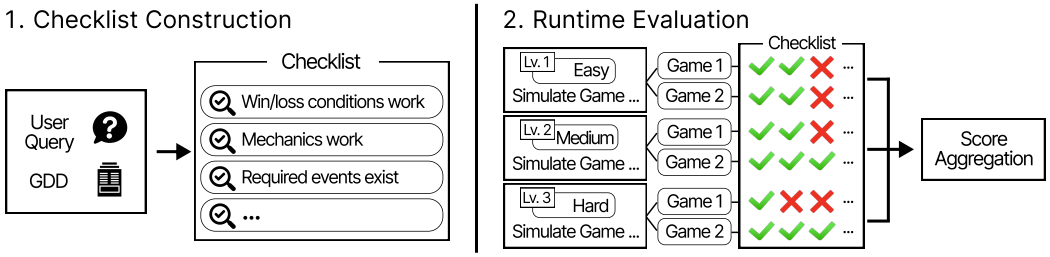


Figure 13: **Behavioral fidelity evaluation.** A checklist of expected behaviors is built from the user query and GDD, and each game is played twice at each difficulty level. A judge checks each item against the log of every rollout, and an item passes if it is observed in any rollout; the score is the fraction of passing items.

and map/zones, checking whether the game starts with the specified units, placements, and zones; *mechanics*, with two items per rule required by the GDD, one checking whether the rule fires at the intended time and the other whether it takes its intended effect; the *victory* and *defeat* specifications, checking whether the game is won or lost when the corresponding condition holds; and the *pacing timeline*, with up to three items checking whether events expected at points on the timeline occur. For example, for a game that restricts the player to static defenses, the mechanics items check whether the production restriction is in force from the start and whether the player's construction units are indeed unable to build factories or air units, and a pacing item checks whether the first enemy wave arrives through the western ramp at around 30 s.

Rollouts. Each game is played twice at each of the three difficulty levels, for six rollouts in total, all run in parallel. The rollouts are played by RTSGAMEAGENT (Sec. 4) with replanning and memory enabled and Gemini-3.8-Flash as its backbone, playing to win. Covering all levels is necessary to exercise the whole checklist: victory is typically observed at easier levels, whereas defeat may only be reached at harder ones. Conversely, if defeat is never observed even at Lv.3, the defeat item fails, so the checklist also reveals games that remain too easy even at their hardest level. Aborted games have no difficulty levels, so their latest script $\mathcal{G}$, which may be incomplete, is played six times to keep the number of rollouts equal across games; behaviors that are already implemented can still be verified even if other parts of the game do not work.

Privileged logging. Since the rollouts serve evaluation rather than play, we log the full simulator state at every step rather than only what the agent observes, including the positions and states of all units of every team, even those outside the agent's field of view, together with engine logs such as rule-script outputs and Lua errors. Logging continues until the game ends or crashes; a crash truncates the log, leaving the behaviors expected afterwards unobserved.

Checklist matching. A judge, Gemini-3.8-Flash, examines the log of each rollout separately and determines, for each checklist item, whether the expected behavior is found. An item passes if it is found in at least one of the six rollouts, regardless of difficulty level, and fails if it never occurs as expected; the score of a game is the fraction of its items that pass. The score therefore measures whether each expected behavior is realized by the implemented game, not how consistently it recurs across rollouts. Although the judge shares its backbone with the generation agents, both conditions compared in Sec. 5.3, *i.e.*, generation with and without experience-driven evolution, use the same backbone, so any self-preference of the judge applies equally to both.

H.2.3 TOKEN USAGE

Token usage measures the cost of producing a game as the total number of input and output tokens consumed by all agents in the framework, from receiving a user query until it outputs a game, including all stages, rollbacks, and difficulty scaling. Unlike wall-clock time, token counts do not depend on API latency or hardware, so costs remain comparable across generations. Token usage is measured for every generation, including aborted ones, whose count covers all calls until the abort; since aborted generations keep running until the rollback budget is exhausted, they tend to consume more tokens, so aborts also raise the average. We report the mean over the five generations in each

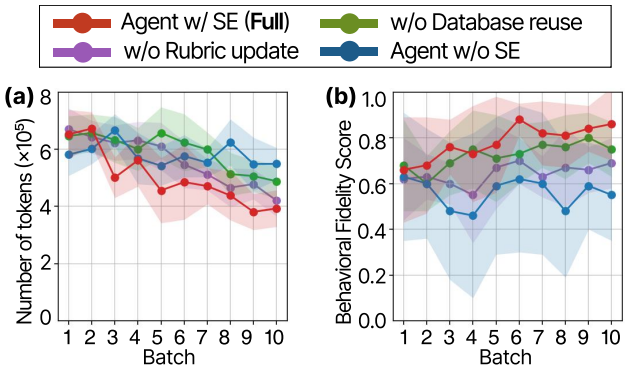


Figure 14: **Ablation of self-evolution mechanisms.** (a) Token usage and (b) behavioral fidelity across ten batches when removing database reuse, rubric updates, or both from *Agent w/ SE*. Shaded areas denote standard deviation.

batch, together with its standard deviation. Experience-driven evolution can reduce this cost in two ways: later queries can fast-track to Stage 4 or reuse verified rule implementations, and refined rubrics help avoid recurring mistakes and the rollbacks they cause. This criterion thus reflects how much accumulated experience reduces redundant work and repeated failures.

H.2.4 Human Preference

Human preference assesses the generated games as a whole, beyond the individual behaviors covered by the checklist. We conduct a blind pairwise study with four RTS-experienced raters, none involved in the benchmark design. From each of the ten batches, we sample two queries for which generation completed both with and without experience-driven evolution, yielding 20 game pairs. Each pair is presented side by side in random order without revealing the source, and every rater plays both games at Lv.2. Since aborted generations, which occur more often without evolution, are excluded, the comparison is, if anything, conservative for the evolved games.

For each pair, raters judge which game is better, or declare a tie, on three criteria: *query alignment*, whether the game realizes the requested scenario, with the described units, objectives, and threats present and the decision the query intends to assess actually arising in play; *fun and balance*, whether the game is enjoyable and fairly challenging, with difficulty neither trivial nor impossible, outcomes attributable to the player's decisions, and sustained tension; and *completeness*, whether the game is functionally well built, running without errors, triggering rules and win/loss conditions as intended, and forming a coherent whole of map, units, and rules with understandable objectives. Each rater's final preference for a pair is the game that wins more criteria, or a tie if both win equally many. For each batch, we report the win, tie, and loss rates of the evolved game over eight judgments (four raters on two pairs).

Inter-rater agreement, measured by Krippendorff's α (Krippendorff, 2011) with win $>$ tie $>$ loss as an ordinal scale, is 0.62 for query alignment, 0.51 for fun and balance, 0.59 for completeness, and 0.58 for the final preference, indicating moderate agreement. Fun and balance, the most subjective criterion, shows the lowest agreement.

H.2.5 Implementation of the End-to-End Baselines

The end-to-end (E2E) baselines generate a complete game in a single LLM call instead of the staged pipeline, using the same backbone (Gemini-3.8-Flash). Along with the user query, this call receives the reference information available to the pipeline: (i) a manual that concatenates the role prompts for GDD writing, Lua rule writing, and game configuration, specifying the format and writing rules of each output; (ii) twelve candidate maps relevant to the query, with their terrain information; (iii) a catalog mapping Armada unit names to their in-game IDs; and (iv) examples of how configurations differ across difficulty levels. As with *Agent w/o SE*, the baselines receive no rubrics, stored rules, or previous generations, and the prompt states that the rule catalog is empty, so every mechanic must be implemented from scratch. The call outputs a GDD, one Lua rule script per mechanic, and game configurations for the three difficulty levels. The GDD plays the same role as in the pipeline; in particular, the behavioral fidelity checklist is derived from it (Appendix H.2.2).

E2E applies only the framework's non-LLM checks, *i.e.*, rule syntax checking and configuration validation, before storing the result, so each generation uses exactly one LLM call. *E2E w/ verify* additionally plays one game in the BAR engine. If the run shows Lua errors, a crash, or a rule that never fires, the logs are fed back and all outputs are regenerated, up to three times, for at most four LLM calls per generation.

H.3 Ablation of Self-Evolution Mechanisms

We ablate the two self-evolution mechanisms of the framework (§3.3) on the same ten batches of five queries, measuring token usage and behavioral fidelity as in §5.3. Without database reuse, validated GDDs and rule sets are not reused, so every query passes through all stages and implements every rule anew; without rubric updates, the retrospective analysis is disabled and the rubrics remain at their initial state. As shown in Fig. 14, *Agent w/ SE* reaches the lowest token usage and the highest behavioral fidelity in later batches, whereas *Agent w/o SE* shows neither trend. Each single-mechanism variant retains part of the gains, with token usage still declining and fidelity still rising, but less than with *Agent w/ SE*. The two mechanisms also appear to contribute differently. Removing database reuse leaves the highest token usage among the single-mechanism variants, suggesting that reuse mainly reduces cost by bypassing redundant stages, whereas removing rubric updates yields lower fidelity, suggesting that refined rubrics mainly improve quality by preventing recurring design mistakes. As the standard deviations overlap in several batches, we regard this distinction as indicative rather than conclusive.

H.4 Examples of Self-Evolving Game Generation

Procedure for game generation. Fig. 22 illustrates one generation cycle of Algorithm 2. The user requests a scenario in which a fragile builder must be escorted to an allied outpost and protected while it works, testing whether the agent can balance keeping it close with clearing or avoiding threats. In Stage 1, the designer turns the query into a scenario brief specifying the escort force, map, and targeted strategic demands; finding no matching GDD in the database, the PM routes generation to Stage 2 instead of fast-tracking. In Stage 2, the designer writes a GDD with the objective, three mechanics, team rosters, and end conditions, together with a checklist for the final game, and the GDD passes the retrieved rubrics, *e.g.*, one requiring defeat conditions in escort scenarios to track allied losses. The PM then retrieves verified implementations of two mechanics from the database, so in Stage 3 the developer writes and tests only the remaining rule, which disables radar to enforce line-of-sight reconnaissance. In Stage 4, the game script passes the rubric check and checklist tests and is scaled to three difficulty levels (omitted from the figure). Finally, the self-evolution phase stores the new GDD, the new rule, and a new rubric for escort-and-fortify scenarios in the database.

Rule and rubric updates during self-evolution. Fig. 23 shows how self-evolution updates the knowledge database. Newly validated rules are committed for reuse, and through the retrospective analysis, the PM traces the failures in the feedback log to their root causes and distills the lessons into rubrics. These updates either add new criteria that guard against the observed failure mode or refine existing ones that proved imprecise, and each is placed under the relevant category of the stage's hierarchy, *e.g.*, end-condition configuration or assault game design. Since later cycles retrieve these rubrics before validation, a recurring design mistake is checked for rather than rediscovered.

Generated diagnostic games. Figs. 24–28 present three representative mini-games generated for each of the five strategic demands. Each example shows the free-form user query, the strategic demand ratings (0–5) recorded in its GDD, the resulting game scenario, and screenshots sampled from a full play-through. The queries state only what the user wants to assess, and the framework grounds each into a concrete scenario with its own map, forces, objectives, and end conditions. In every example, the focal demand receives the highest rating while the others are rated lower, mirroring the design of the human-designed mini-games (Sec. 3.2). Scenarios sharing the same focal demand also differ substantially, *e.g.*, in whether the player acts alone or alongside an allied AI and whether it attacks, defends, or escorts, showing that the framework produces functionally diverse scenarios from free-form queries.

Table 15: **Agent evaluation on generated games.** WR of RTSGAMEAGENT with two backbones on one generated game per strategic demand, over five runs per level.

		Gemini-3.7-Flash			GLM-5.3-Flash		
Game	Focal demand	Lv.1	Lv.2	Lv.3	Lv.1	Lv.2	Lv.3
EUP	Resource management	0.80	0.20	0.00	0.80	0.40	0.00
WLE	Spatial & temporal reasoning	0.60	0.40	0.00	0.40	0.40	0.00
RFD	Opponent modeling	0.80	0.20	0.00	0.60	0.20	0.00
AFC	Collaboration	1.00	0.40	0.00	0.40	0.20	0.00
TRA	Adversarial planning	0.80	0.20	0.00	0.20	0.00	0.00

H.5 AGENT EVALUATION ON GENERATED GAMES

As a preliminary check of generated games in agent evaluation, we evaluate RTSGAMEAGENT with Gemini-3.7-Flash and GLM-5.3-Flash on five generated games, each created using the query from the top example in one of Figs. 24–28. The five games are Economy Under Pressure (EUP, resource management), Wounded Leader Escape (WLE, spatial and temporal reasoning), Reserve Force Diversion (RFD, opponent modeling), Allied Front Coordination (AFC, collaboration), and Threat Response Allocation (TRA, adversarial planning), each played five times per level and evaluated by WR (Tab. 15). For both backbones, WR never increases with the level, indicating that difficulty scaling yields graded difficulty. No run wins at Lv.3, suggesting that it may be harder than Lv.3 of the human-designed mini-games, although five runs cannot rule out occasional wins. Gemini-3.7-Flash matches or exceeds GLM-5.3-Flash in every game except EUP, where GLM-5.3-Flash wins one more game at Lv.2, and the gap is largest on AFC and TRA. This pattern loosely mirrors the human-designed mini-games, where GLM-5.3-Flash is competitive on TCP but falls behind on ATS and AWD (Tab. 4), although the small number of games and runs limits this comparison.

System Prompt

```
You are managing a memory system for a game AI agent.
Your task is to maintain a concise list of experiences based on existing memories and new triggers, and to retrieve relevant memories for the current situation.

You will receive:
1. Current experiences: Previously stored insights and lessons
2. New triggers: Recent game events that may contain valuable information

Your job is to:
1. Update the memory by maintaining the FINAL list of experiences that should be retained
2. Retrieve relevant memories that are useful for the current decision-making

For memory updates, you may:
- Keep existing experiences that are still relevant
- Remove outdated or redundant experiences
- Add new experiences derived from triggers
- Merge similar experiences into one

For each experience in update_memory, output:
- "content": A single sentence summarizing the insight
- "tag": A descriptive tag you create (e.g., fact, inference, hypothesis, outcome, lesson, ongoing_operation, etc.)
- "time": The game time when this insight was observed

For retrieve_memory, select experiences from the updated memory that are most relevant to the current game situation and decision-making.

Trigger types:
- decision_outcome: Previous decision's intent vs actual outcome (builds completed, squads arrived, units produced)
- battle_outcome: Deaths/damage clustered per engagement, with win/loss/trade/skirmish label
- enemy_observation: Changes in enemy sightings — first_seen, lost_sight, advancing paths
- self_status: Anomalies and milestones only (resource issues, commander threat, army-size thresholds)

Focus on retaining:
1. Enemy movement patterns and compositions
2. Battle outcomes and their causes
3. Strategic decisions and their results
4. Territorial control changes

Output format:
{
  "update_memory": [
    {"content": "experience sentence", "tag": "category_tag", "time": game_time},
    ...
  ],
  "retrieve_memory": [
    {"content": "experience sentence", "tag": "category_tag", "time": game_time},
    ...
  ]
}

Output empty arrays for either key if no experiences are worth retaining or retrieving.
```

Figure 15: **System prompt for the memory phase.** The system prompt used in the memory phase to update the agent's memory and retrieve relevant experiences before the decision phase.

Input Example


```
{
  "current_experiences": [
    {
      "content": "Enemy static economy and defenses were sighted clustered around (84-90,91-97), including a Solar Collector at (84,96),
                  a Metal Extractor at (86,96), and Sentries at (84,97) and (90,91).",
      "tag": "enemy_base", "time": 179
    },
    {
      "content": "Enemy expansion at (60,89) is established/active with at least a Metal Extractor and Solar Collector present and
                  periodic Construction Vehicle activity nearby.",
      "tag": "enemy_expansion", "time": 1081
    },
    {
      "content": "Enemy raid pressure often includes a Blitz+Whistler pairing rather than a lone skirmisher.",
      "tag": "enemy_composition", "time": 309
    }
  ],

  "new_triggers": [
    {
      "content": "Decision outcome 17:24→18:00",
      "metadata": {
        "type": "decision_outcome",
        "prev_decision_time": 1044,
        "current_time": 1080,
        "elapsed": 36,
        "build_outcomes": [
          {"intent": "Metal Extractor", "target_pos": [22, 88], "assigned_builders": [45231], "status": "completed"},
          {"intent": "Sentry",         "target_pos": [30, 82], "assigned_builders": [45231], "status": "pending_or_failed"}
        ],
        "production_outcomes": {
          "queued_total": 6,
          "new_units_seen": {"Rocketeer": 4, "Mace": 2}
        },
        "movement_outcomes": [
          {"squad": "scout_tick",   "target": [26, 92], "current_center": [25, 90], "distance_to_target": 3, "arrived": false, "alive_count": 3},
          {"squad": "main_assault", "target": [30, 82], "current_center": [30, 82], "distance_to_target": 1, "arrived": true,  "alive_count": 5}
        ]
      }
    },
    {
      "content": "Battle near minimap [35,83] 17:26→17:35",
      "metadata": {
        "type": "battle_outcome",
        "time_window": [1046, 1055],
        "center_pos": [35, 83],
        "allied_losses": {"Rocketeer": 2, "Mace": 1},
        "enemy_losses":  {"Blitz": 1, "Whistler": 1},
        "allied_damage_taken": 1240,
        "key_losses": [],
        "outcome": "trade"
      }
    },
    {
      "content": "Enemy observation 17:24→18:00",
      "metadata": {
        "type": "enemy_observation",
        "time_window": [1044, 1080],
        "events": [
          {"event": "first_seen",  "name": "Blitz",    "pos": [35, 83], "cell": [7, 16], "at": 1045},
          {"event": "first_seen",  "name": "Whistler", "pos": [42, 83], "cell": [8, 16], "at": 1046},
          {"event": "advancing",   "name": "Blitz",    "path_cells": [[7, 16], [6, 16]], "window": [1045, 1055]},
          {"event": "lost_sight",  "name": "Construction Vehicle", "cell": [12, 17]}
        ],
        "current_composition": {"Blitz": 1, "Whistler": 1}
      }
    },
    {
      "content": "Self status 17:24→18:00",
      "metadata": {
        "type": "self_status",
        "time_window": [1044, 1080],
        "resource_anomalies": [
          {"type": "metal_capped",         "duration_sec": 25},
          {"type": "energy_income_change", "delta": 42.5}
        ],
        "commander_change": {
          "hp_now": 0.87,
          "hp_delta": -0.09,
          "pos": [12, 88],
          "nearest_enemy_distance": 18,
          "threat": "moderate"
        },
        "milestones": [
          {"event": "army_size_100", "at": 1068, "value": 103}
        ]
      }
    }
  ]
}
```


Figure 16: **Input example for the memory phase.** The input to the memory phase consists of the currently retained experiences and newly emitted triggers since the last decision — including decision outcomes, battle outcomes, enemy observations, and self-status changes.

Output Example


```
{
  "update_memory": [
    {
      "content": "Enemy static economy and defenses were sighted clustered around (84-90,91-97), including a Solar Collector at (84,96),
                  a Metal Extractor at (86,96), and Sentries at (84,97) and (90,91).",
      "tag": "enemy_base",
      "time": 179
    },
    {
      "content": "Enemy expansion at (60,89) is established/active with at least a Metal Extractor and Solar Collector present and
                  periodic Construction Vehicle activity nearby.",
      "tag": "enemy_expansion",
      "time": 1081
    },
    {
      "content": "Enemy raid comp of Blitz+Whistler has been re-confirmed at (35-42,83) — same pairing observed at 05:09 continues
                  to be the standard raid unit.",
      "tag": "enemy_composition",
      "time": 1046
    },
    {
      "content": "Traded 2 Rocketeer + 1 Mace for 1 Blitz + 1 Whistler at (35,83) around 17:26-17:35, taking 1240 damage —
                  Blitz+Whistler duo is roughly one-for-one against our current front line.",
      "tag": "battle_outcome",
      "time": 1055
    },
    {
      "content": "main_assault squad (5 alive) reached and holds (30,82) — front-line anchor established in the mid-north approach.",
      "tag": "squad_status",
      "time": 1080
    },
    {
      "content": "Sentry construction at (30,82) failed within the 36s decision window despite an assigned builder — area may be contested or builder
                  is under fire; re-attempt with escort.",
      "tag": "build_failed",
      "time": 1080
    },
    {
      "content": "Enemy Blitz advancing westward from cell (7,16) toward (6,16) — likely probing our (30,82) position from the east.",
      "tag": "enemy_movement",
      "time": 1055
    },
    {
      "content": "Commander HP dropped -9% to 0.87 with nearest enemy at distance 18 (moderate threat) around 18:00 — needs defensive attention.",
      "tag": "threat",
      "time": 1080
    },
    {
      "content": "Metal capped for ~25s within the 17:24-18:00 window while energy income spiked (+42.5) — spending capacity is underutilized,
                   need more factories or higher production queue.",
      "tag": "economy",
      "time": 1080
    },
    {
      "content": "Army size crossed 100 units at 17:48 — mid-game force threshold reached.",
      "tag": "milestone",
      "time": 1068
    }
  ],
  "retrieve_memory": [
    {
      "content": "Commander HP dropped -9% to 0.87 with nearest enemy at distance 18 (moderate threat) around 18:00 — needs defensive attention.",
      "tag": "threat",
      "time": 1080
    },
    {
      "content": "Metal capped for ~25s within the 17:24-18:00 window while energy income spiked (+42.5) — spending capacity is underutilized,
                  need more factories or higher production queue.",
      "tag": "economy",
      "time": 1080
    },
    {
      "content": "Sentry construction at (30,82) failed within the 36s decision window despite an assigned builder — area may be contested or builder
                  is under fire; re-attempt with escort.",
      "tag": "build_failed",
      "time": 1080
    },
    {
      "content": "Enemy Blitz advancing westward from cell (7,16) toward (6,16) — likely probing our (30,82) position from the east.",
      "tag": "enemy_movement",
      "time": 1055
    },
    {
      "content": "Enemy raid comp of Blitz+Whistler has been re-confirmed at (35-42,83) — same pairing observed at 05:09 continues to be
                   the standard raid unit.",
      "tag": "enemy_composition",
      "time": 1046
    }
  ]
}
```


Figure 17: **Output example for the memory phase.** The output of the memory phase consists of an updated memory state that incorporates the new triggers and a subset of those experiences retrieved for the upcoming decision.

System Prompt

```
You are an RTS game AI commander for Beyond All Reason.
Your task is to:
1. Create an optimal build order for construction and assign builders to each construction
2. Determine the unit production queue and the builder assist allocation for each factory
3. Organize all military units into tactical squads and command each squad with a multi-step trajectory.

Description: {{GAME_DESCRIPTION}}
Knowledge: {{GAME_KNOWLEDGE}}

IMAGE INPUT FORMAT:
- First image: Minimap view of the entire battlefield
- Second image: Screen view (on-screen area, centered at the camera position specified below)
- {{TEAM_COLORS}}
- ALWAYS use 0-100 grid system for ALL coordinates

TEXT INPUT FORMAT:
{
    "world": {
        "time": Game time in MM:SS format,  "wind": Current wind speed,
        "resource_spots": {"metal": [[x, y], ...], "geo": [[x, y], ...]},
        "resources": {
            "metal":  {"income": float, "expense": float, "current": int, "max": int},  "energy": {"income": float, "expense": float, "current": int, "max": int}
        } },
    "groups": Dict of squads → list of {"id", "name", "position", "health"}. Special key "unassigned" holds unattached units.
    "builders": List of {"id", "name", "position", "current_task"}.
    "buildings": Dict of completed non-factory friendly buildings → list of [x, y] positions.
    "factories": Dict of completed friendly factories → list of {"id", "position"}.
    "under_construction": List of {"id", "name", "position", "eta_seconds", "builders"}.
    "damaged_buildings": List of {"id", "name", "position", "health", "eta_seconds", "builders"}.
    "enemies": Dict of enemies → {"position", "detected_by": "sight"|"radar"}.
    "allies_N": (Team games) Dict of allied team N's units → {"position", "health"}.
    "memory": {"experience": list of {content, tag, time},  "strategy": (optional) map of key → {description}}
}

OUTPUT FORMAT:
Response: <JSON object with the following structure>
{
  "build_construction": [ ["building_name", [position_x, position_y], [builder_id, ...]],  [building_id, [builder_id, ...]], ... ],
  "unit_production": { "factory_id": ["unit_name", "unit_name", ...], ... },
  "factory_assist": { "factory_id": [builder_id, ...], ... },
  "groups": { "squad_name": [unit_id, unit_id, ...], ... },
  "group_movement": {
    "squad_name": {
      "steps": [
        {
          "idx": <int>,    "command": "move" | "attack" | "stop",    "position": [x, y],
          "on_event": [
            {"if": <condition>, "goto": <idx_of_target_step>},
            ... ] }, ... ] }, ...
  },
  "reason": "Explain your reasoning for construction, production, and squad organization."
}

OUTPUT FORMAT DETAILS:
- Do not include any annotation in your output. It can't be decoded!

[Build Construction]
1. Start new: ["building_name", [x, y], [builder_id, ...]]
   - building_name: human-readable name from unit knowledge base (e.g. "Metal Extractor"). Codes like "armmex" are dropped.
   - builder_id list may be empty (any idle builder picks up); same builder repeated = sequential build.
2. Assist under-construction: [building_id, [builder_id, ...]]
   - Prefer assisting a stalled site over duplicating a structure.
- Queue only what assigned builders can start/progress within {{INFERENCE_INTERVAL}}s.

[Unit Production]
- factory_id → ordered production queue. Names must be human-readable (e.g. "Construction Bot").
- Queue only as many as the factory can produce within {{INFERENCE_INTERVAL}}s.

[Factory Assist]
- factory_id → builder_id list to guard/assist. Optional; omit or pass {} when no assist needed.

[Groups]
- Overwrites entire squad organization every inference. Every combat unit you want to command must appear in exactly one squad.
- Do NOT include Armada Commander or Construction Bots — filtered server-side.
- Prefer multiple distributed squads over one giant stack.

[Group Movement]
- Each squad's trajectory is a list of steps. Each step has its own `idx` (unique within the squad); `goto` values reference these `idx`s. Execution starts at `idx: 0`.
- Every tick, engine walks the current step's `on_event` list in order; first match wins and squad jumps to `goto` target.
- `on_event` is optional. A step without `on_event` is terminal — holds command until next LLM inference.
- `goto` can go backward (loop), forward (skip), or to any valid `idx` — arbitrary state machines including cycles.
- Next replan in ~{{INFERENCE_INTERVAL}}s. Set condition `seconds`/`timeout` relative to that budget.
- Command semantics:
  - "attack": Move to "position" and engage enemies encountered along the way.
  - "move": Move to "position" ignoring enemies on the path.
  - "stop": Hold current position. Takes no position.
- All coordinates in 0-100 grid.

[Condition Primitives — "if" values inside on_event]
Every `if` MUST be one of the following four object shapes. String expressions like `"elapsed > 15"` are rejected.
- {"type": "idle_for", "seconds": <int>}
    Squad has been idle for `seconds` continuously (natural "step complete" signal — arrives after movement finishes / combat resolves).
- {"type": "timeout", "seconds": <int>}
    `seconds` elapsed since this step started (blunt escape hatch for stuck squads).
- {"type": "hp_below", "percent": <int>}
    Squad average HP fraction * 100 < `percent`. Use for retreat/regroup branches.
- {"type": "group_size_below", "count": <int>}
    Fewer than `count` units alive. Use for depletion-triggered fallbacks.
```

Figure 18: **System prompt for the decision phase.** Placeholders (denoted with double braces {{...}}) in the system prompt are replaced with the game description, unit knowledge, team color specification, and inference interval.

Input Example

Textual Observation


```
{
  "world": {
    "time": "19:24",  "wind": 18.47,
    "resource_spots": {
      "metal": [[80, 4], [12, 5], [39, 10], [56, 22], [72, 45]], "geo": []
    },
    "resources": {
      "metal":  {"income": 24.3, "expense": 18.1, "current": 1250, "max": 2000}, "energy": {"income": 187.5, "expense": 152.0, "current": 3400, "max": 5000}
    }
  },
  "groups": {
    "scout_tick": [
      {"id": 3733,  "name": "Tick", "position": [52, 78], "health": 1.00},
      {"id": 15876, "name": "Tick", "position": [33, 62], "health": 0.83},
      {"id": 20542, "name": "Tick", "position": [32, 48], "health": 1.00}
    ],
    "home_guard": [
      {"id": 10864, "name": "Mace", "position": [37, 66], "health": 1.00},
      {"id": 7418,  "name": "Mace", "position": [38, 64], "health": 1.00},
      {"id": 28580, "name": "Mace", "position": [37, 64], "health": 1.00}
    ],
    "main_assault": [
      {"id": 13756, "name": "Rocketeer", "position": [34, 40], "health": 1.00},
      {"id": 7675,  "name": "Mace",     "position": [39, 32], "health": 0.88},
      {"id": 132,   "name": "Mace",     "position": [40, 30], "health": 0.84}
    ],
    "unassigned": []
  },
  "builders": [
    {"id": 4501, "name": "Construction Bot", "position": [14, 12], "current_task": "build:Advanced Bot Lab (34%)"},
    {"id": 4502, "name": "Construction Bot", "position": [12, 10], "current_task": "assist:8801"},
    {"id": 4503, "name": "Construction Bot", "position": [13, 9],  "current_task": "idle"}
  ],
  "buildings": {
    "Metal Extractor": [[13, 6], [6, 6], [14, 21]],
    "Solar Collector": [[12, 8], [13, 8], [15, 8]],
    "Wind Turbine":    [[8, 8], [7, 8], [10, 8]]
  },
  "factories": {
    "Bot Lab": [{"id": 8801, "position": [14, 12]}]
  },
  "under_construction": [
    {"id": 8802, "name": "Advanced Bot Lab", "position": [18, 14], "eta_seconds": 42,       "builders": [4501]},
    {"id": 8803, "name": "Fusion Reactor",   "position": [10, 15], "eta_seconds": "stalled", "builders": []}
  ],
  "damaged_buildings": [
    {"id": 8750, "name": "Metal Extractor", "position": [6, 6], "health": 0.62, "eta_seconds": 8, "builders": [4502]}
  ],
  "enemies": {
    "Metal Extractor": [{"position": [60, 89], "detected_by": "sight"}],
    "Solar Collector": [{"position": [60, 87], "detected_by": "sight"}],
    "Blitz":           [{"position": [65, 78], "detected_by": "sight"}],
    "unknown":         [{"position": [70, 55], "detected_by": "radar"}]
  },
  "memory": {
    "experience": [
      {"content": "Enemy expansion at (60,89) is established with Mex + Solar and periodic Construction Vehicle activity.",
       "tag": "enemy_expansion", "time": 1081},
      {"content": "Enemy raid comp of Blitz+Whistler has been re-confirmed at (35-42,83) — same pairing observed earlier.",
       "tag": "enemy_composition", "time": 1046},
      {"content": "Traded 2 Rocketeer + 1 Mace for 1 Blitz + 1 Whistler at (35,83) around 17:26-17:35 — roughly one-for-one against the current front line.",
       "tag": "battle_outcome", "time": 1055},
      {"content": "Sentry construction at (30,82) failed within the 36s decision window despite an assigned builder — area may be contested.",
       "tag": "build_failed", "time": 1080},
      {"content": "Metal capped for ~25s while energy income spiked (+42.5) — spending capacity is underutilized, need more factories.",
       "tag": "economy", "time": 1080}
    ]
  }
}
```


Visual Observation

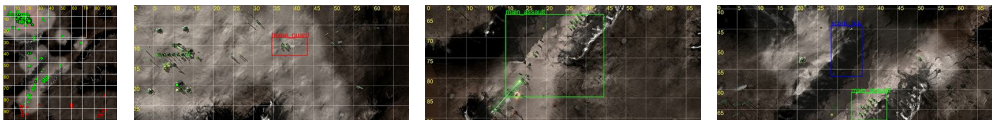


Figure 19: **Input example for the decision phase.** The input to the decision phase consists of textual and visual observations of the current game state, together with experiences retrieved by the memory phase.

Output Example

```
{
  "build_construction": [
    ["Metal Extractor", [14, 21], [4503]],
    [8802, [4501, 4503]],
    [8750, [4502]],
    ["Solar Collector", [16, 8], []],
    ["Wind Turbine", [11, 8], []],
    ["Sentry", [32, 42], []],
    ["Advanced Radar Tower", [15, 15], []]
  ],
  "unit_production": {
    "8801": ["Rocketeer", "Rocketeer", "Mace", "Construction Bot"]
  },
  "factory_assist": {
    "8801": [4502]
  },
  "groups": {
    "scout_tick":   [3733, 15876, 20542],
    "home_guard":   [10864, 7418, 28580],
    "main_assault": [13756, 7675, 132]
  },
  "group_movement": {
    "scout_tick": {
      "steps": [
        {
          "idx": 0,
          "command": "move",
          "position": [34, 87],
          "on_event": [
            {"if": {"type": "idle_for", "seconds": 5},   "goto": 1},
            {"if": {"type": "hp_below", "percent": 40},  "goto": 2}
          ]
        },
        {
          "idx": 1,
          "command": "move",
          "position": [60, 89],
          "on_event": [
            {"if": {"type": "timeout", "seconds": 30}, "goto": 2}
          ]
        },
        {
          "idx": 2,
          "command": "move",
          "position": [20, 20]
        }
      ]
    },
    "home_guard": {
      "steps": [
        {
          "idx": 0,
          "command": "stop",
          "on_event": [
            {"if": {"type": "group_size_below", "count": 10}, "goto": 1}
          ]
        },
        {
          "idx": 1,
          "command": "move",
          "position": [15, 12]
        }
      ]
    },
    "main_assault": {
      "steps": [
        {
          "idx": 0,
          "command": "attack",
          "position": [46, 45],
          "on_event": [
            {"if": {"type": "idle_for", "seconds": 4},  "goto": 1},
            {"if": {"type": "hp_below", "percent": 50}, "goto": 2}
          ]
        },
        {
          "idx": 1,
          "command": "attack",
          "position": [60, 89]
        },
        {
          "idx": 2,
          "command": "move",
          "position": [25, 30]
        }
      ]
    }
  },
}
```

Figure 20: **Output example for the decision phase.** The output of the decision phase specifies actions across five decision categories: constructing buildings, producing units, assigning builders to assist factories, organizing units into squads, and specifying multi-step trajectories for each squad.

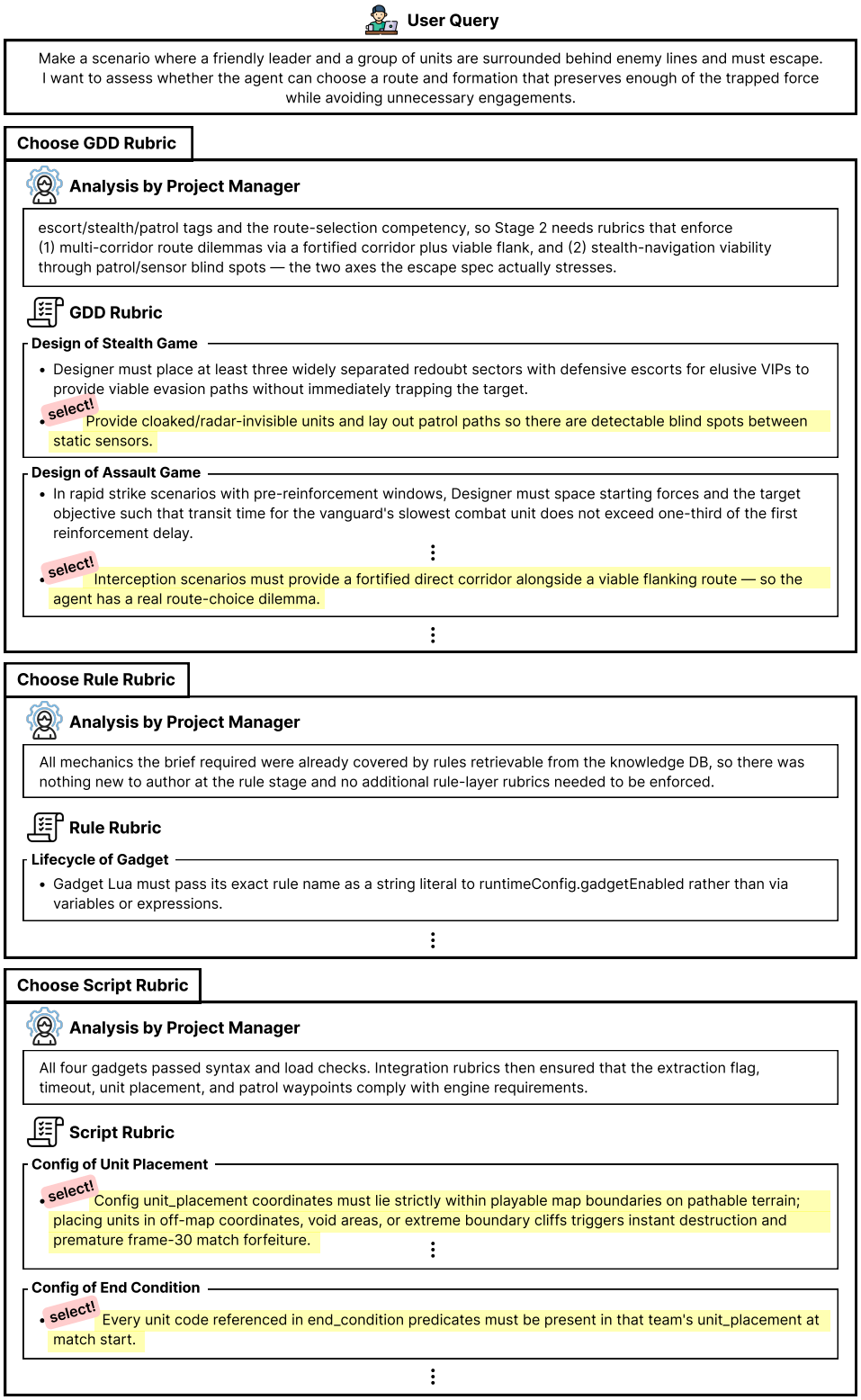


Figure 21: **Rubric selection by the Project Manager.** For a user query, project manager chooses scope-appropriate rubrics at three authoring stages — GDD, Rule, and Script — and provides a short rationale for each selection.

Stage 1 - Scenario Brief Generation		
User	Ask Query	Make a scenario where a fragile builder must be escorted through dangerous territory to a friendly outpost and protected while it works. I want to assess whether the agent can balance keeping the builder close to protection with clearing or avoiding threats along the route.
Designer	Generate Scenario Brief	**Brief** The player commands a mixed escort detachment (skirmishers and raiders) and a single high-value, slow-moving, fragile construction unit starting at a staging area. Across the rugged terrain of Avalanche .... **Map** Avalanche 3.4 **Demands** spatial-temporal reasoning : 5, opponent modeling : 5, ...
Project Manager	Retrieve GDD from DB	None
	Retrieve Rubric for Stage 2	Rubric A : In multi-team escort or ally assault scenarios, defeat conditions tracking allied loss must require .... Rubric B : When evaluating an agent's ability to distribute defensive coverage across uncertain attack ....
	Routing	Continue to stage 2
Stage 2 - GDD Generation		
Designer	Generate GDD	**Objective** Escort a fragile Construction Bot and Commander alongside an armed security detachment from a northern staging staging area through hostile patrol-infested corridors to reach a vulnerable allied ... **Mechanics** *'disable_radar_tech' :* Enforces strict line-of-sight reconnaissance under Fog of War by preventing the .... *'outpost_patrol_circuit' :* Controls scheduled cyclical enemy patrol squadrons that loop between waypoints .... *'converging_wave_assault' :* Triggers progressive enemy ground attack waves that spawn from three outer .... **Rosters** Team 1 (Player): 1 Armada Commander at (750, 950), 1 Construction Bot at (720, 960), 3 Blitz at (780, 930) .... Team 2 (Allied Outpost): 1 Armada Commander at (3100, 2400), 1 Solar Collector at (3050, 2420), ... Team 3 (Enemy Forces): 2 Pawn at (1900, 1500), 2 Rocketeer at (2300, 2100), 1 Solar Collector at .... **End conditions** victory: game time reaches 180 seconds AND team 1 has at least 1 Construction Bot unit AND team 2 has .... defeat: team 1 has 0 Armada Commander units OR team 1 has 0 Construction Bot units OR team 2 has 0 ... **Balance notes** ....
	Generate Checklist for Stage 4	Checklist A : At frame 1, the game is not over; neither the victory nor the defeat condition is satisfied yet. Checklist B : Sustained combat occurs across the scenario: at least 10 damage events are recorded ... Checklist C : The scenario completes within 5500 frames without engine hang or indefinite stall, cleanly ....
Analyst	Rubric Check	✓ Rubric A ✓ Rubric B
Project Manager	Retrieve Rule from DB	outpost_patrol_circuit's code converging_wave_assault's code
	Retrieve rubric for stage 3	Rubric A : Gadget Lua must pass its exact rule name as a string literal to runtimeConfig.gadgetEnabled .... Rubric B : Config unit_placement coordinates must lie strictly within playable map boundaries on pathable ....
	Routing	Continue to stage 3
Stage 3 - Rule Set Construction		
Developer	Generate rule set	Rule code 'disable_radar_tech' Test code 'disable_radar_tech'
Analyst	Rubric & Code Check	**Rubric check** ✓ Rubric A ✓ Rubric B **Code check** Test code passed
Project Manager	Retrieve rubric for stage 4	Rubric A : Every unit code referenced in end_condition predicates must be present in that team's unit .... Rubric B : In end_condition ['or', ...], every separate loss or victory trigger must be an isolated single ....
	Routing	Continue to stage 4
Stage 4 - Game Finalization		
Developer	Generate Script	{ "information": { "description": "Escort your fragile Construction Bot and Armada Commander alongside an armed .... "decision": ["construction", "movement"], "match_format": "2v1", "time_limit": 180, ...
Analyst	Rubric check	✓ Rubric A ✓ Rubric B
	Checklist Test	✓ Checklist A ✓ Checklist B ✓ Checklist C ✓ Checklist D
Project Manager	Routing	Accept
	Self-Evolution	**Add GDD** : Radar-blind builder escort **Add rule** : disable_radar_tech **Add rubric** : In escort-and-fortify scenarios, Designer must delay destination assault waves past the ....

Figure 22: **Example of the game generation process.**

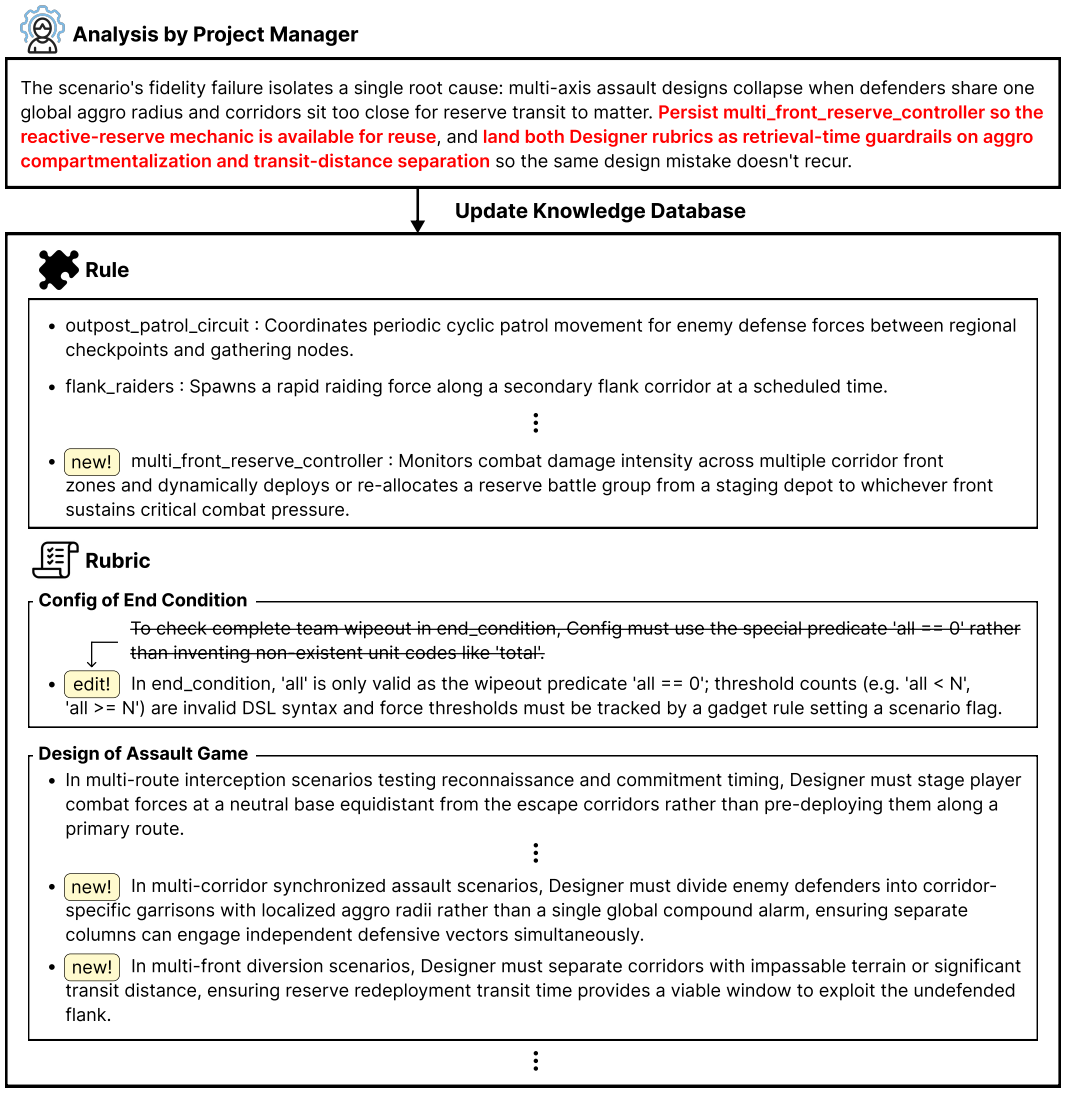


Figure 23: **Example of rule and rubric updates during Self-Evolving.** From the Project Manager's judgment highlighted in **red bold text**, reusable mechanics are appended to the rule catalog for later scenarios, and new rubrics are added to reduce the recurrence of the observed failure mode.

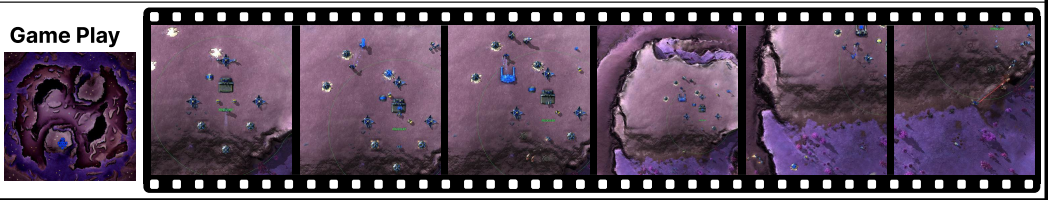
Generated diagnostic example (Focal Strategic Demand : Resource management)

User Query: Make a scenario where the player has a growing resource income and several ways to invest it while enemy pressure gradually increases. I want to assess whether the agent can decide when to reinvest in the economy, when to expand production, and when to save resources for immediate military strength.

Strategic Demands: Resource management (5), Spatial&temporal reasoning (3), Opponent modeling (2), Collaboration (0), Adversarial planning (3)

Game Scenario: Defend the southern redoubt against escalating hostile assault waves across multiple converging corridors. The player must strategically invest their initial resource base and expanding economy—deciding when to expand metal and energy extraction, when to build auxiliary factory capacity, and when to mass mobile combat forces or fortify chokepoints. Victory is achieved by surviving for 300 seconds (or eliminating all assault waves), while defeat occurs immediately if the player's Commander is destroyed.

Game Play



User Query: Make a scenario where defensive structures must be placed along elevated terrain overlooking multiple enemy approaches. I want to assess whether the agent can decide how to distribute limited defensive coverage across routes that cannot all be protected equally.

Strategic Demands: Resource management (5), Spatial&temporal reasoning (4), Opponent modeling (2), Collaboration (0), Adversarial planning (3)

Game Scenario: Command an elevated redoubt atop the central plateau of Hotstepper against escalating enemy assault columns converging from three lowland corridors (west, east, and south). The player must distribute finite construction resources and micro a small mobile reserve to establish static defenses along high-ground choke points overlooking the ramps. Defeat occurs if the primary Command Center (Decoy Fusion Reactor) or Commander on the central plateau is destroyed; victory is achieved by repelling all three converging assault waves and surviving until the hostile offensive collapses at 180 seconds.

Game Play



User Query: Make a scenario where several resource fields are available with different levels of safety, distance, and remaining value. I want to assess whether the agent can decide where to send harvesters and when shifting workers to another resource area is worth the disruption.

Strategic Demands: Resource management (5), Spatial&temporal reasoning (3), Opponent modeling (2), Collaboration (0), Adversarial planning (2)

Game Scenario: The player command must coordinate a workforce of harvesters and construction units across three distinct resource salvage zones (a safe, low-yield backyard field; an intermediate midfield plateau; and an exposed, metal-rich central graveyard) while managing hostile patrol patterns. The player achieves victory by gathering 3500 metal from field reclamation and extraction to reach the production quota before 180 seconds elapse (or sustaining field operations with the Armada Commander and harvesters intact when the 180-second clock expires). Defeat occurs if the player's Armada Commander is destroyed or if all harvester units are lost before the quota is secured.

Game Play

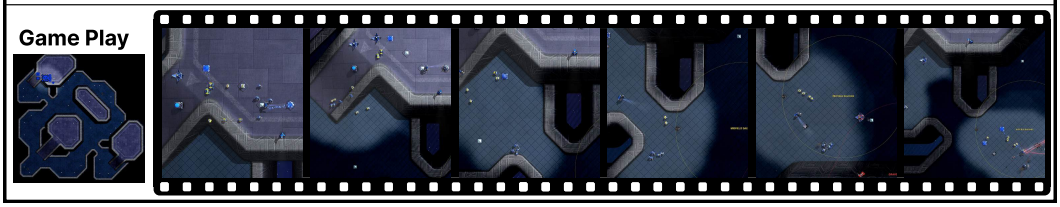


Figure 24: **A mini-game focused on resource management, generated by the self-evolving framework.** Each example shows the free-form user query, the strategic demands the generated scenario targets, the resulting game scenario, and screenshots sampled from a full play-through.

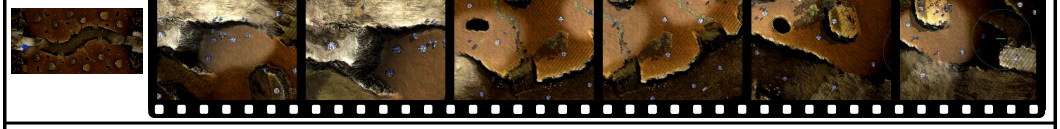
Generated diagnostic example (Focal Strategic Demand : Spatial&temporal reasoning)

User Query: Make a scenario where a wounded friendly leader must be escorted away from enemy pursuers to safety. I want to assess whether the agent can choose a route and decide when escorts should delay the pursuers instead of remaining close to the vulnerable leader.

Strategic Demands: Resource management (1), Spatial&temporal reasoning (5), Opponent modeling (4), Collaboration (2), Adversarial planning (4)

Game Scenario: The player commands a damaged Armada Commander who must escape hostile pursuers across the Ditched_V1 terrain and reach the northeastern extraction landing zone. The enemy deploys relentless mobile pursuit waves behind the player. To survive, the player must evaluate the terrain, navigate through viable corridors, and sacrifice or position escort defenders at narrow chokepoints to stall pursuers. Victory is achieved when the Armada Commander successfully reaches the extraction zone. Defeat occurs if the Armada Commander is destroyed or if the mission timer exceeds 180 seconds before reaching extraction.

Game Play

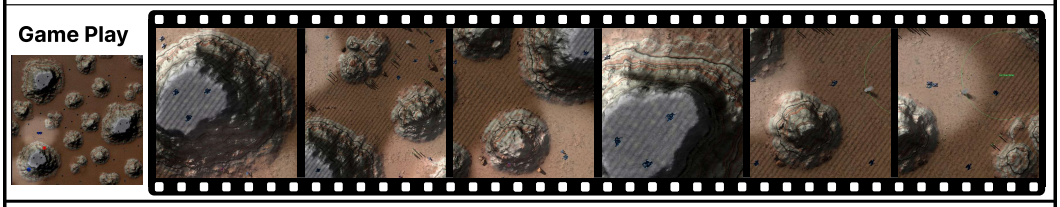


User Query: Make a scenario where a small stealth force must cross an area patrolled by enemy units without being detected. I want to assess whether the agent can observe patrol movement and decide when to move, hide, or change routes.

Strategic Demands: Resource management (0), Spatial&temporal reasoning (5), Opponent modeling (4), Collaboration (0), Adversarial planning (3)

Game Scenario: The player commands a small team of light infiltration units starting at one edge of the map, tasked with traversing to an extraction zone on the opposite side. The enemy force consists of heavier combat patrols moving on predetermined looping patrol paths through the central lowland and side corridors between the prominent mesas. Gameplay centers on timing movement windows, holding position behind mesa high-ground to break line of sight, and selecting detour routes when primary lanes are blocked by overlapping patrols. Detection prompts nearby patrols to break path and aggressively hunt down the infiltrators, making evasion and patient observation critical to survival.

Game Play



User Query: Make a scenario where long-range player units must dismantle a network of enemy fixed defenses. I want to assess whether the agent can decide where to position and reposition the attackers so they can exploit their range without exposing themselves to counter-fire.

Strategic Demands: Resource management (1), Spatial&temporal reasoning (5), Opponent modeling (2), Collaboration (0), Adversarial planning (3)

Game Scenario: The player commands a focused task force of mobile long-range strike units (such as artillery crawlers and long-range skirmishers) starting in the outer approach corridor. The opposing team holds an integrated network of static defensive bastions spread across Digsite's central basin, ramps, and elevated rim chokepoints, composed of short-range anti-swarm turrets, medium-range beam/plasma emplacements, and key infrastructure anchors that create nested threat envelopes.

Game Play

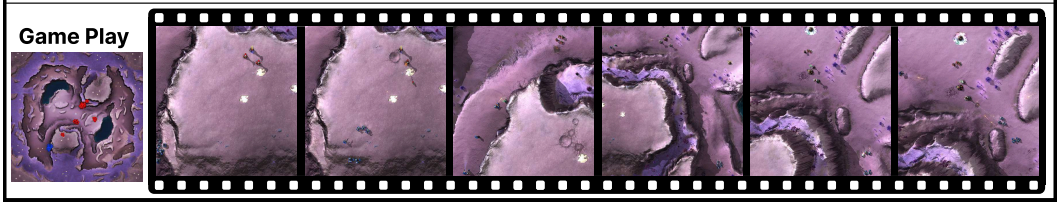


Figure 25: **A mini-game focused on spatial & temporal reasoning, generated by the self-evolving framework.** Each example shows the free-form user query, the strategic demands the generated scenario targets, the resulting game scenario, and screenshots sampled from a full play-through.

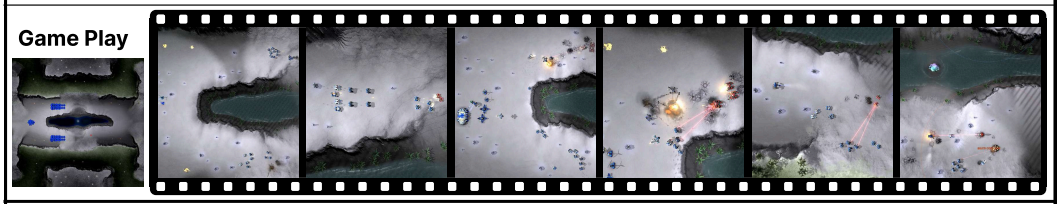
Generated diagnostic example (Focal Strategic Demand : Opponent modeling)

User Query: Make a scenario where the enemy has a limited reserve force that shifts toward whichever front is under the most pressure. I want to assess whether the agent can decide how to apply pressure across multiple fronts so the enemy's reserve movement can be exploited.

Strategic Demands: Resource management (1), Spatial&temporal reasoning (3), Opponent modeling (5), Collaboration (0), Adversarial planning (3)

Game Scenario: The player commands a mobile strike force starting on the western approaches, facing an entrenched enemy defending high-value infrastructure (command center and energy complex) in the eastern base. Vittra's long central lake bifurcates the map into isolated northern and southern corridors, separated by impassable water and rugged bluffs that impose a substantial transit delay for ground units redeploying between lanes.

Game Play

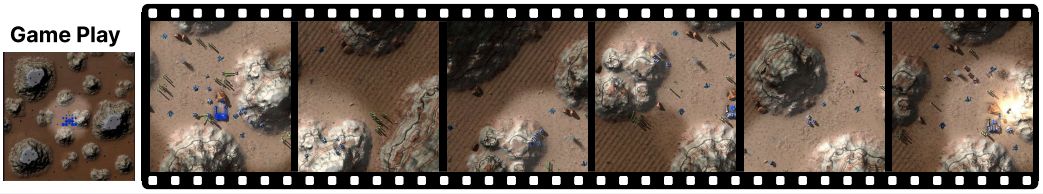


User Query: Make a scenario where an enemy VIP attempts to escape through one of several routes and reacts to where the player concentrates forces. I want to assess whether the agent can decide how much information to gather and how long to delay committing to a particular interception route.

Strategic Demands: Resource management (0), Spatial&temporal reasoning (3), Opponent modeling (5), Collaboration (0), Adversarial planning (4)

Game Scenario: The player controls a mobile interception detachment comprising fast reconnaissance units (radar/scout rovers) and a mid-weight strike force (raiders/assault skirmishers), starting from a central staging area. The enemy side consists of an unarmed or lightly defended high-tech VIP transport escorted by a small vanguard, spawning in a secluded staging zone and seeking to reach any of three extraction points located at opposite map edges through three distinct valley corridors around the central mesas.

Game Play



User Query: Make a scenario where a fragile builder must be escorted through dangerous territory to a friendly outpost and protected while it works. I want to assess whether the agent can balance keeping the builder close to protection with clearing or avoiding threats along the route.

Strategic Demands: Resource management (1), Spatial&temporal reasoning (4), Opponent modeling (5), Collaboration (2), Adversarial planning (3)

Game Scenario: The player commands a mixed escort detachment (skirmishers and raiders) and a single high-value, slow-moving, fragile construction unit starting at a staging area. Across the rugged terrain of Avalanche 3.4, an allied forward outpost has sustained damage and requires the builder to construct an energy grid link or defensive anchor to secure the sector. Hostile enemy forces hold key chokepoints along the lake-edge corridors, fielding roving patrols and static ambush positions.

Game Play

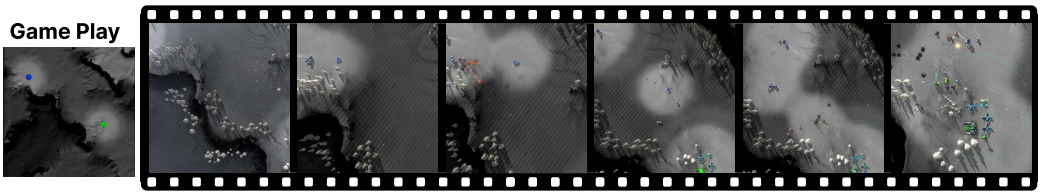


Figure 26: **A mini-game focused on opponent modeling, generated by the self-evolving framework.** Each example shows the free-form user query, the strategic demands the generated scenario targets, the resulting game scenario, and screenshots sampled from a full play-through.

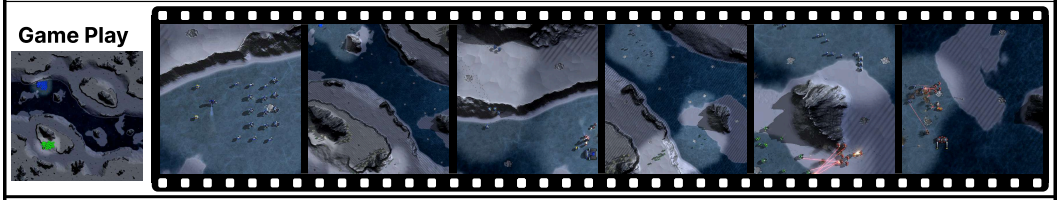
Generated diagnostic example (Focal Strategic Demand : Collaboration)

User Query: Make a scenario where an allied AI holds one front while the player pushes into enemy territory from another. I want to assess whether the agent can balance forward pressure with the need to stay flexible enough to support the ally if the situation changes.

Strategic Demands: Resource management (1), Spatial&temporal reasoning (3), Opponent modeling (2), Collaboration (5), Adversarial planning (2)

Game Scenario: The scenario places the Player and an Allied AI on one side of the central water divide, facing an entrenched Enemy force across two primary crossing corridors. The Allied AI commands a heavily fortified defensive position anchoring the southern shore crossing, while the Player controls a flexible ground assault task force positioned to push through the northern corridor toward the enemy's main industrial base. The Enemy mounts sustained pressure against the ally while defending the northern approach with static emplacements and reactionary patrols.

Game Play

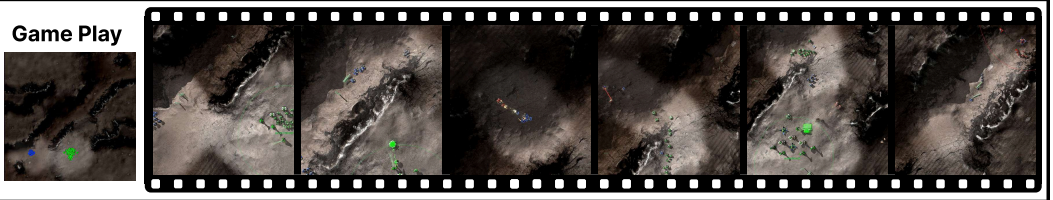


User Query: Make a scenario where an allied AI protects the main base while the player raids the enemy economy. I want to assess whether the agent can decide how deeply to commit to the raid while judging whether continuing the attack is worth the growing pressure on the ally.

Strategic Demands: Resource management (2), Spatial&temporal reasoning (3), Opponent modeling (2), Collaboration (5), Adversarial planning (3)

Game Scenario: The player and an allied AI share a defensive posture against an aggressive enemy AI. The allied AI commands a fortified base along one main river crossing lane, fielding durable frontline units and static defenses to absorb periodic, intensifying enemy assault waves. The player commands a fast, mobile raiding detachment operating along the opposite flank lane, targeting dispersed enemy metal extractors, energy converters, and vulnerable tech structures.

Game Play



User Query: Make a scenario where a bridge must be secured so friendly forces can move through a contested area. I want to assess whether the agent can decide how much attention to commit to controlling the chokepoint while other threats try to draw forces away.

Strategic Demands: Resource management (1), Spatial&temporal reasoning (4), Opponent modeling (2), Collaboration (5), Adversarial planning (2)

Game Scenario: The player controls a tactical defense force deployed near the central ditch crossing on Ditched_V1. Autonomous allied convoy units spawn at regular intervals on one side of the ditch and must cross the narrow land bridge to reach a safe evacuation zone on the opposite side. The enemy deploys mixed forces: steady frontline pushes directly toward the crossing point to overwhelm the bridgehead, combined with mobile skirmishers and flank detachments that attack peripheral structures and radar stations to lure the player away. Play tests whether the agent maintains sufficient defensive mass at the crossing during allied transits while using mobile strike elements to check flanking diversions without leaving the primary objective exposed.

Game Play

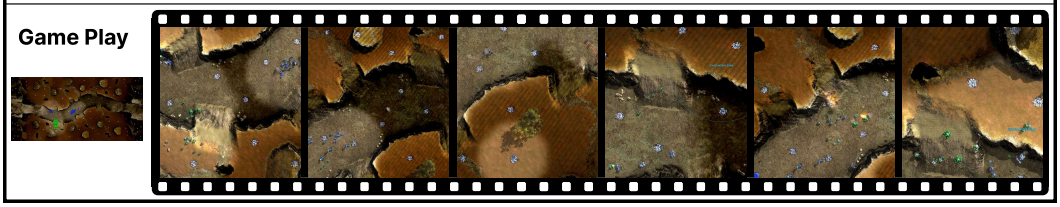


Figure 27: **A mini-game focused on collaboration, generated by the self-evolving framework.** Each example shows the free-form user query, the strategic demands the generated scenario targets, the resulting game scenario, and screenshots sampled from a full play-through.

Generated diagnostic example (Focal Strategic Demand : Adversarial planning)

User Query: Make a scenario where a powerful enemy unit is advancing toward the player's base. I want to assess whether the agent can decide when and where to concentrate enough firepower to stop it without leaving other areas exposed.

Strategic Demands: Resource management (2), Spatial&temporal reasoning (4), Opponent modeling (2), Collaboration (0), Adversarial planning (5)

Game Scenario: The player holds a fortified forward base on the south end of Copper Hill, combining a static defensive belt of heavy and light laser turrets with a mixed mobile force of assault mechs, warriors, and rocketeers. After a preparation window, a slow, heavily armored enemy Behemoth — flanked by a small heavy-tank escort and dedicated anti-air — pushes south along the central corridor toward the player's Commander. The scenario asks the player to use the setup time to concentrate defensive firepower along the expected axis and time focus fire against the escorted juggernaut once it enters engagement range, keeping the Commander alive until the assault column is broken.

Game Play

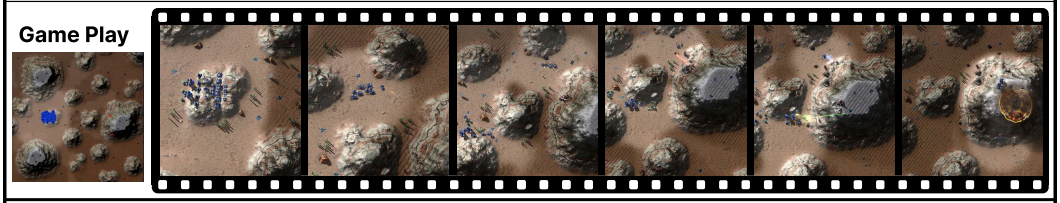


User Query: Make a scenario where the player must penetrate a fortified enemy position and reach a high-value leader inside. I want to assess whether the agent can decide where to create an opening and how to spend combat strength while moving through layers of defensive resistance.

Strategic Demands: Resource management (2), Spatial&temporal reasoning (3), Opponent modeling (4), Collaboration (0), Adversarial planning (5)

Game Scenario: The player commands a potent combined-arms tactical strike group consisting of assault tanks, mobile artillery, and screening skirmishers, starting in the open lowlands. The enemy team occupies a fortified high-ground redoubt centered on one of the mesa plateaus, protected by outer picket bunkers, layered static defense rings (heavy turrets, radar, screen shields), and localized response patrols guarding distinct ramp approaches. Deep inside sits the high-value enemy commander unit.

Game Play



User Query: Make a scenario where several small player squads begin scattered across the battlefield and need to combine their strength. I want to assess whether the agent can decide which groups should move toward each other, where they should meet, and when it is worth acting before the entire force has assembled.

Strategic Demands: Resource management (1), Spatial&temporal reasoning (4), Opponent modeling (3), Collaboration (0), Adversarial planning (5)

Game Scenario: The player commands 3-4 small, specialized skirmish squads (e.g., light assault raiders, skirmishers, and support units) spawned in separate quadrants of Tetrad_V2. The enemy holds a fortified base in the center or a fortified quadrant, while sending roving hunter patrols through the lowland passes that threaten isolated groups. To succeed, the player must navigate the terrain passes, avoid superior hostile patrols, and coordinate routes so that squads link up safely. Secondary enemy targets (such as vulnerable power nodes or isolated convoys) appear across the map on timers, tempting the agent to strike opportunistically with localized partial forces before final assembly against the primary redoubt.

Game Play

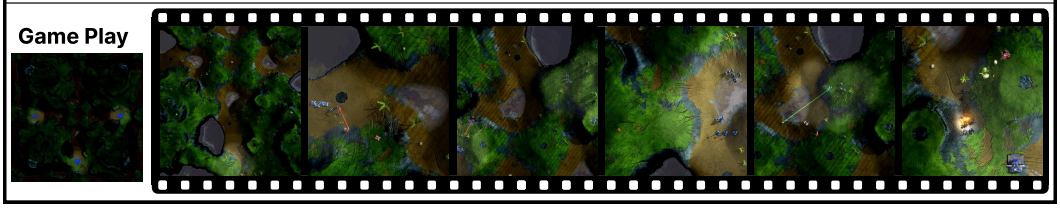


Figure 28: **A mini-game focused on adversarial planning, generated by the self-evolving framework.** Each example shows the free-form user query, the strategic demands the generated scenario targets, the resulting game scenario, and screenshots sampled from a full play-through.