GAMEHATCH3D: AN END-TO-END AGENTIC FRAMEWORK FOR 3D GAME GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Traditional 3D game development requires substantial human effort in asset creation, programming, and playtesting. Recent agentic frameworks have shown promise in automating the planning, execution, and revision of complex systems. However, end-to-end automated 3D game generation remains at an early stage and presents additional challenges. It requires agents to jointly construct complex scenes and characters, model physical behavior, and implement gameplay logic. The resulting combinatorial complexity increases reasoning overhead and reduces generation reliability. To address these challenges, we propose GameHatch3D, an end-to-end agentic framework for generating 3D games from a text prompt. GameHatch3D first extracts game-design cues from the input prompt and consolidates them into a structured game specification. Guided by this specification, HatchCraft, our game-construction harness, coordinates reusable Environment, Character, Physics, and Rule Skills to construct the core components of a functional game. These skills encode domain-specific construction knowledge, enabling more reliable and efficient game construction. To support automated refinement, we further equip the agent with an engine-guided playtesting mechanism that enables dual-path self-correction. Engine-provided state information guides navigation planning and action generation, while playtesting observations and failures are used to diagnose visual and gameplay defects. For evaluation, we establish a benchmark comprising diverse gameplay demonstrations and curated high-level game descriptions, together with a comprehensive evaluation protocol. Experiments show that GameHatch3D generates higher-quality and more playable games than a general-purpose agentic baseline while consuming fewer tokens.

1 INTRODUCTION

3D video games offer rich spatial structures and flexible forms of interaction, enabling players to explore and shape immersive virtual worlds. Beyond entertainment, these environments can also serve as scalable data sources for research in world modeling (Valevski et al., 2025; Li et al., 2025; Che et al., 2025) and embodied AI (Yang et al., 2024b; Xia et al., 2026; Roberts et al., 2026). However, bringing a game idea to life remains a complex and resource-intensive process. Even a structurally simple game often requires close collaboration among professionals from multiple disciplines, posing a high barrier to entry. Making game creation more affordable and accessible has therefore long been a shared goal of academia and industry.

Recent advances in large language models (LLMs) and vision-language models (VLMs) have demonstrated broad capabilities in reasoning and generation, reshaping software engineering and creative production (Zhao et al., 2023; Jimenez et al., 2024; Naveed et al., 2025). In agentic systems, these models serve as backends, while agent harnesses extend their capabilities through automated workflows for planning, execution, and revision, making the generation of complex systems increasingly feasible (Agashe et al., 2024; Pham et al., 2026). This progress points toward the more ambitious goal of generating complete games from high-level descriptions through an autonomous pipeline. Game generation, however, imposes requirements beyond functional correctness and executability. A game may run successfully yet remain unsatisfactory due to poor visuals, implausible physical interactions, or inconsistent rules. Meeting these requirements calls for an agent specialized for game creation.

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.

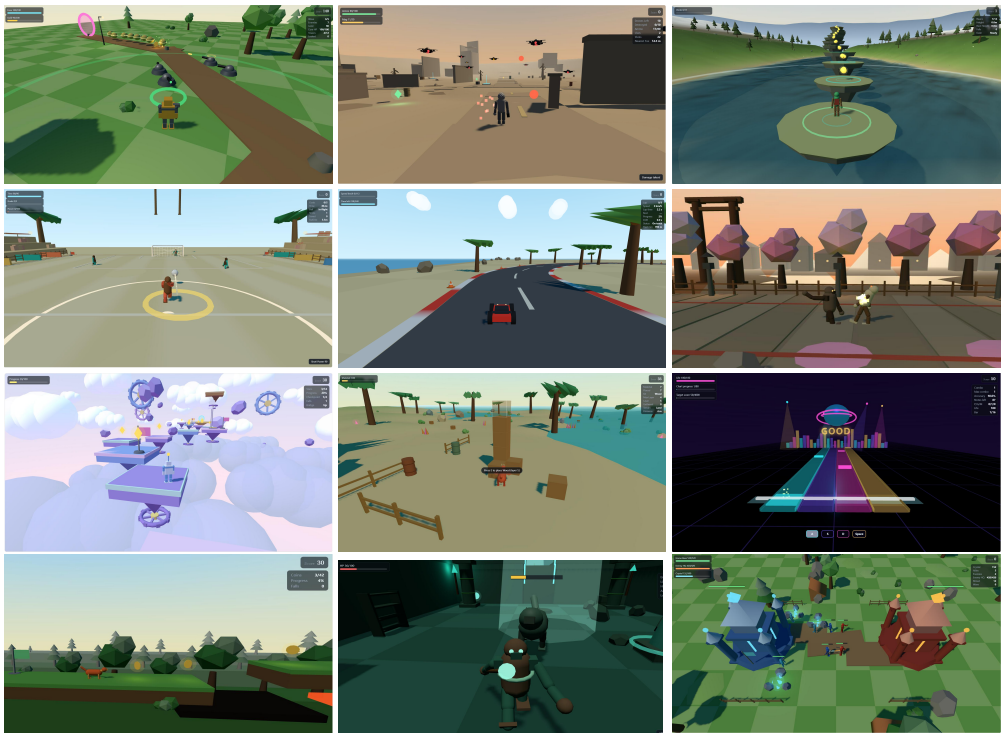


Figure 1: GameHatch3D generates diverse and playable browser-based 3D games from text prompts. It supports a wide range of game genres while consuming fewer tokens than a general-purpose agentic baseline using the same backend model.

End-to-end agentic game creation remains at an early stage (Zheng et al., 2026; Bolton et al., 2025). Recent work such as OpenGame (Jiang et al., 2026) demonstrates the feasibility of translating high-level prompts into playable 2D web games through structured project scaffolding, coordinated code generation, and compilation-guided debugging. Automated 3D game creation, however, presents additional challenges. It involves building spatially coherent environments that integrate terrain, architecture, and environmental objects, while rendering these elements with high-quality materials, visual effects, and lighting. The increased spatial complexity expands the range of possible gameplay behaviors, complicating physical interactions and game rule design. General-purpose agents face higher reasoning costs and greater generation uncertainty in this broader design space. Generated games often require repeated human intervention to improve quality and playability.

In this paper, we introduce GameHatch3D, an end-to-end agentic framework for creating 3D games from text prompts. The input prompt specifies high-level game-design requirements. The backend model parses these requirements and consolidates them into a structured game specification. Based on this specification, **HatchCraft**, our game-construction harness, coordinates four reusable domain-specific skills that collectively form the foundation of a functional game. The **Environment Skill** defines the spatial layout and visual composition of the scene; the **Character Skill** constructs character appearance, motion, and control; the **Physics Skill** establishes physically consistent interactions between characters and the environment; and the **Rule Skill** organizes gameplay objectives and conditions into executable state-transition logic. HatchCraft manages structured information exchange and cross-component updates among these skills to produce a coherent and executable game. These skills provide the backend model with reusable construction priors, reducing the reasoning overhead and token consumption required to build each game from scratch. In addition, we equip the agent with an engine-guided automated playtesting mechanism that supports dual-path self-correction. The mechanism combines precise engine-provided state information with the backend model's high-level planning and diagnosis capabilities, enabling automated gameplay execution and the detection of both visual and gameplay defects. The detected defects are then converted into refinement signals for iterative improvements in visual quality and playability. To evaluate the gener-

ated games, we further introduce a benchmark comprising diverse gameplay demonstrations paired with curated high-level game descriptions. We also provide a comprehensive evaluation protocol that measures physical correctness, playability and visual quality. Experiments show that GameHatch3D outperforms general-purpose agentic baselines across multiple quality dimensions while consuming fewer tokens. The main contributions are summarized as follows:

- We introduce **GameHatch3D**, an end-to-end agentic framework for generating browser-based 3D games from text prompts.
- We develop **HatchCraft**, a game-construction harness that coordinates four domain-specific skills, improving generation reliability while reducing reasoning overhead and token consumption.
- We equip the agent with an engine-guided automated playtesting mechanism that supports dual-path self-correction for iterative detection and repair of visual and gameplay defects.
- We establish a benchmark and comprehensive evaluation protocol for 3D game generation.

2 Related Work

2.1 General-Purpose Agentic Frameworks

Large language models (LLMs) and vision-language models (VLMs) have demonstrated strong capabilities in understanding, multimodal perception, and complex reasoning (Chan et al., 2025; Zhang et al., 2024). However, directly applying foundation models remains unreliable for long-horizon tasks that require persistent state tracking, coordination across interdependent subtasks, and repeated verification. Agentic frameworks improve reliability through closed-loop workflows that iteratively execute actions and refine decisions from feedback (Yao et al., 2023; Shinn et al., 2023). This paradigm has also been extended to repository-level software engineering, where agents modify code and repair failures using execution results (Jimenez et al., 2024; Yang et al., 2024a). Despite their generality, existing agentic frameworks are not directly suited to end-to-end game creation. Unlike conventional software tasks that primarily emphasize functional correctness, game generation must jointly maintain visual coherence, physical plausibility, and consistent gameplay. A game may therefore compile and launch while remaining visually poor or mechanically incomplete, motivating specialized harnesses that incorporate game-development priors and evaluate generated games through actual rendering and interaction.

2.2 Agentic Systems for Gaming

Existing work on game-oriented agents has largely focused on action control for automated gameplay (Magne et al., 2026; Yue et al., 2025; Karten et al., 2026; Wu et al., 2026), while agent-driven game generation is still at an early stage. OpenGame (Jiang et al., 2026) generates playable 2D web games through reusable templates and execution-guided debugging. In the 3D domain, prior methods have mainly aimed to construct executable prototypes through structured workflows in Unity (Unity Technologies, 2005) or Unreal Engine (Epic Games, 1998). UniGen (Yang et al., 2025) employs specialized agents for primitive-first scene construction and debugging in Unity, although its evaluation is limited to three game prototypes. DreamGarden (Earle et al., 2025) introduces hierarchical planning for user-steerable prototyping in Unreal Engine, primarily targeting simulations with a single viewpoint and no player interaction. AutoUE (Yin et al., 2026) constructs scenes using retrieved assets and procedural content generation, with offline playtesting used for evaluation. In contrast, GameHatch3D explores open-ended programmatic 3D game creation in Three.js. Its reusable environment, character, physics, and rule skills support coordinated construction and revision across interdependent game components. The proposed automated playtesting mechanism leverages engine states for navigation and action generation, while using visual observations and subgoal outcomes to diagnose and correct visual and gameplay defects. Our work offers a new perspective on agentic 3D game creation by shifting the focus from structured prototype assembly to the open-ended construction and automated refinement of playable games.

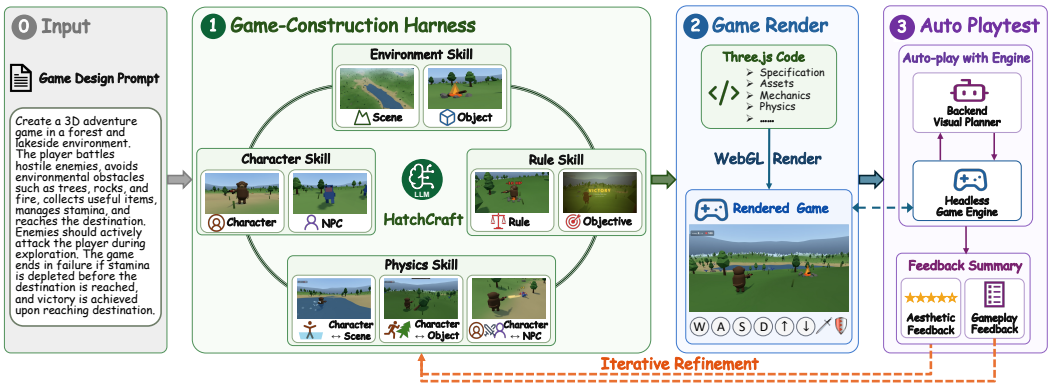


Figure 2: Overview of the GameHatch3D framework. Given a text prompt, HatchCraft coordinates specialized Environment, Character, Physics, and Rule Skills to construct a browser-based 3D game. The resulting Three.js implementation is rendered with WebGL. The generated game is then refined through automated playtesting to improve visual quality and playability.

3 METHOD

3.1 OVERVIEW

GameHatch3D is an end-to-end agentic framework for 3D game generation built on Three.js. It programmatically constructs game scenes, characters, physics, and gameplay logic. Figure 2 illustrates the overall workflow of GameHatch3D. It takes as input a text prompt that specifies high-level design requirements. The backend model then converts the prompt into a structured game specification following a predefined schema. Our agent harness, HatchCraft, then constructs the game prototype in a coarse-to-fine manner according to the specification by invoking four specialized implementation skills that guide component-wise code generation. To address visual and game-design defects exposed during gameplay, we further introduce an engine-guided automated playtesting framework with dual-path self-correction. In a headless rendering environment, the backend model follows a closed-loop decision-and-refinement process. In each playtesting cycle, it combines visual observations with engine-provided physical states to generate the next subgoal and corresponding action rollout. The visual and gameplay feedback is processed through two self-correction paths to improve visual quality and address game-design defects.

3.2 GAME HARNESS WITH DOMAIN-SPECIFIC SKILLS

Programmatic 3D game construction spans multiple specialized domains, including visual content generation, physical modeling, and gameplay-logic design, each with distinct implementation and validation requirements. General-purpose agents typically decompose an input specification into subtasks. Without domain-specific guidance, however, they may resort to relatively coarse solutions for challenging components, particularly visual content generation and physical interaction design. Fig. 3(a)–(c) presents two representative examples from our pilot study, in which we used Cursor with Claude Opus 4.8 (Anthropic, 2026) as the backend model to generate 30 games from textual specifications. These examples reveal several functional limitations. First, collectibles are embedded in platforms and projectiles pass through NPCs, preventing objective completion (Fig. 3(a)). Second, some platforms cannot be reached and characters remain suspended above the water, indicating problems with movement and physical interactions (Fig. 3(b)). Third, falling into the sea does not trigger failure, and standing in fire does not reduce health, showing that the specified hazard responses are missing (Fig. 3(c)). The game environments and character models are also visually simplistic and lack detail. In addition, Cursor consumes substantially more tokens than GameHatch3D across the same 30 game-generation tasks (Fig. 3(d)).

Motivated by these observations, we restructure the game-creation process around four reusable skills: (1) **Environment Skill**, which guides the construction of terrain, buildings, scene objects, materials, and lighting, establishing the spatial layout and visual composition of the game world. It

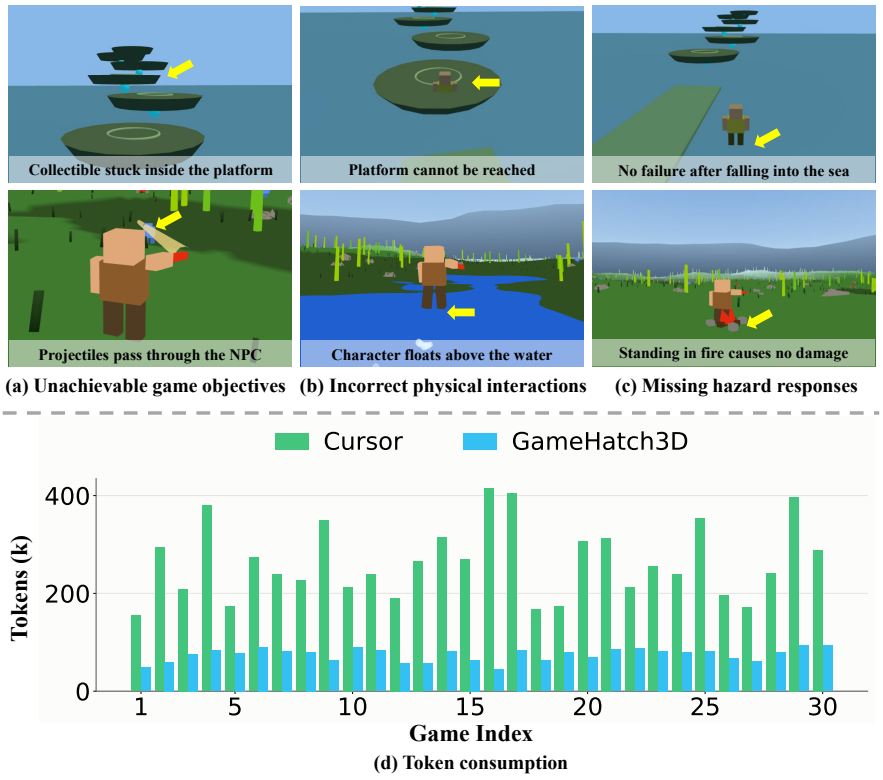


Figure 3: Pilot study of 3D game generation using Cursor with Claude Opus 4.8 (Anthropic, 2026) as the backend model. In (a)–(c), the top and bottom rows show a platforming game and a shooting game, respectively. These examples illustrate unachievable objectives, incorrect physical interactions, and missing hazard responses. (d) Token consumption of Cursor and GameHatch3D across 30 game-generation tasks using the same backend model.

also specifies spatial information, such as ground height and world boundaries, to support character placement and physical interactions; (2) **Character Skill**, which guides the construction of player and non-player characters, including their geometry, part hierarchy, appearance, and animation behavior; (3) **Physics Skill**, which guides the design of character movement and interactions with the environment, including collision handling, contact responses, and movement constraints; and (4) **Rule Skill**, which specifies scoring and objectives, connecting player actions and game events to changes in game state. Together, these skills provide reusable construction guidance for the visual, physical, and gameplay aspects of a game.

To build the skill library, we task the agent with generating a collection of games and iteratively refining and debugging their implementations. We then organize recurring construction procedures, implementation patterns, and failure cases into four reusable skills. Each skill specifies the component's responsibilities, required inputs, and output format, and provides construction guidance and code examples. These skills guide the backend model in generating structured component descriptions from the game specification. The descriptions expose game-specific properties, such as geometry, appearance, motion parameters, and rule thresholds, as configurable fields.

To organize these skills, we introduce HatchCraft, a game harness that coordinates construction according to the dependencies among game components. Given a structured game specification, HatchCraft invokes the required skills in sequence. At each step, the backend model uses the skill guidance, the game specification, and the descriptions of previously constructed components to produce a structured description of the current component. HatchCraft introduces cross-component updates (CCU) to resolve incompatibilities among components by allowing the backend model to propose adjustments to the configurable properties of previously constructed components. For example, when a platform cannot be reached with the current jump height, the physics construction

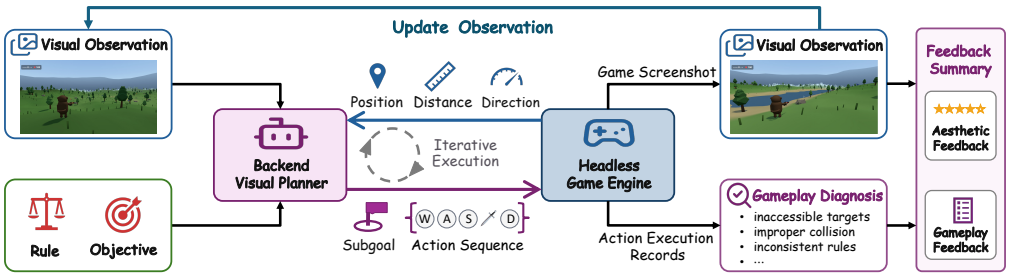


Figure 4: Illustration of engine-guided automated playtesting with dual-path self-correction and refinement. The backend model combines visual observations, the global objective, game rules, and engine-provided physical states to select a subgoal and generate an executable action sequence. The headless game engine executes the actions and returns updated observations and execution records, which serve as aesthetic and gameplay refinement signals, respectively, to improve visual quality and playability.

step may propose increasing the character's jump height, subject to the constraints of the game specification. HatchCraft applies the proposed changes and makes the updated descriptions available to subsequent construction steps. Once construction is complete, a deterministic assembler converts the final descriptions into scene objects, scripts, event connections, and state machines, producing an executable game for automated playtesting and refinement.

3.3 Engine-Guided Automated Playtesting and Refinement

Although HatchCraft can generate runnable games, it remains limited in identifying defects that emerge during actual gameplay. A straightforward approach is to let the backend model control the character through a frame-by-frame loop of visual observation and action prediction. However, repeatedly interpreting rendered frames and making low-level control decisions incurs substantial token consumption and prolongs the generation process. We therefore equip the agent with an engine-guided automated playtesting mechanism featuring dual-path self-correction, as illustrated in Figure 4. The key idea is to use engine-provided states for action execution while reserving the backend model for high-level planning and defect diagnosis. Playtesting relies on a shared runtime interface that exposes game states and objective progress, with objectives and interactive entities registered by the generated rule code. In each playtesting cycle, the backend model combines the current frame, game rules, and engine-provided states to select a subgoal and generate a structured action sequence. Each subgoal, such as collecting an item or reaching a designated location, includes a completion condition that can be checked against engine states. The executor converts the actions into keyboard and mouse inputs in a headless browser, using the player's current position to adjust movement toward target locations.

During playtesting, the backend model inspects sampled frames for visual defects and produces aesthetic feedback. We downsample these frames before model inspection to reduce the token cost of visual processing. For gameplay diagnosis, the model examines execution records, before-and-after states, and rendered frames to assess whether a failed subgoal reflects a game defect or an ineffective action sequence. Potential defects include inaccessible targets, improper collision settings, and inconsistent rules. HatchCraft uses the aesthetic and gameplay feedback to revise the affected game code, then reruns compilation and playtesting to check the changes. Through iterative playtesting, diagnosis, and refinement, this mechanism improves both the visual quality and playability of the generated game.

4 Experiments

4.1 Evaluation Details

Benchmark. For evaluation, we curate 210 gameplay videos from Gaming-500-Hours (Markov AI, 2026) and YouTube, spanning 21 game genres with 10 videos per genre. For each video, we extract a 20-second clip illustrating the core gameplay and use Claude Opus 4.8 to generate a high-level spec-

Backend	Method	Token (K)↓	Runtime (min)↓	Physics↑	Playability↑	Visual Quality↑	Total Score↑
GPT-5.5	Direct	12.1	3.8	23.7	27.1	32.3	27.7
	Cursor	204.8	10.9	50.4	60.5	52.9	54.6
	GameHatch3D	65.9	9.2	72.6	73.5	73.2	73.1
Claude Opus 4.8	Direct	14.2	4.2	25.1	28.9	34.3	29.4
	Cursor	223.2	10.4	50.6	62.1	54.3	55.7
	GameHatch3D	73.3	9.8	74.8	75.5	73.1	74.5

Table 1: Comparison of direct backend generation, Cursor (Anysphere, 2026), and GameHatch3D using GPT-5.5 (OpenAI, 2026) and Claude Opus 4.8 (Anthropic, 2026). *Direct* denotes generation using the backend model without an agentic workflow. Runtime and token consumption are averaged across all generated games.

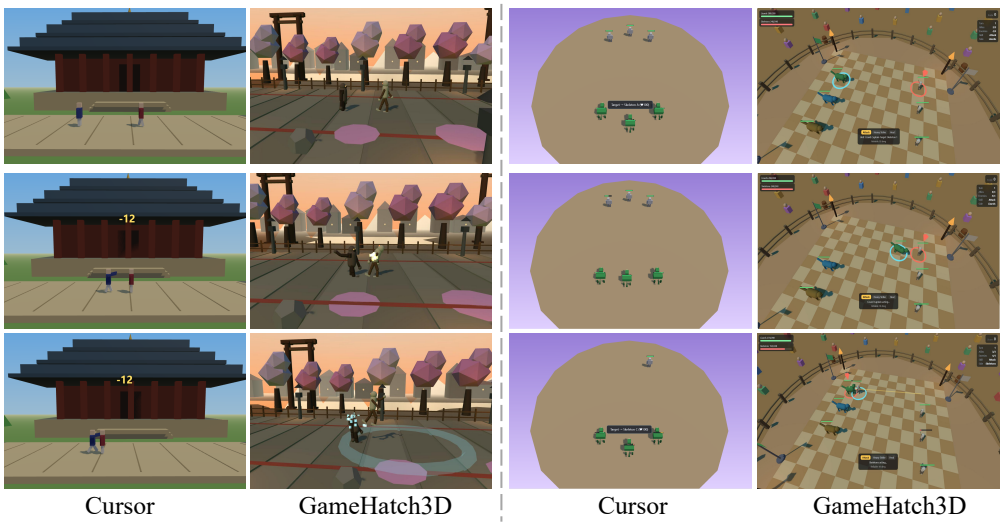


Figure 5: Qualitative comparison between Cursor (Anysphere, 2026) and GameHatch3D for 3D game generation, using Claude Opus 4.8 (Anthropic, 2026) as the backend model. The game-design specifications are: Left, "Two fighters duel in a temple yard in side view until one's health is drained to zero." Right, "A giant-lizard captain and a skeleton squad trade turns in an arena until one side is wiped out."

ification of the environment, characters, interactions, and success and failure conditions. GPT-5.5 checks each specification against the corresponding clip and flags potentially unsupported details, such as character abilities or success conditions not shown in the video. Human reviewers then verify the flagged details against the clip and correct the specification. The resulting specifications serve as text inputs for game generation. Evaluation focuses on the visual quality and playability of the generated games, with the source videos providing gameplay requirements rather than targets for frame-level reconstruction.

Evaluation protocol. We evaluate each generated game along three complementary dimensions. (1) **Physical Correctness.** For each game specification, we define more than 10 physical-interaction tests, such as checking whether characters can stand on platforms without falling through them or move without passing through solid obstacles. The same tests are used across all methods. Runtime states and rendered observations are checked against the expected outcome of each test, with Claude Opus 4.8 assisting visual assessment. Each test receives a score of 1 if the expected behavior is verified and 0 otherwise. The final score is the percentage of predefined tests that pass. (2) **Playability.** An automated controller attempts the specified intermediate subgoals and final objective through player actions. All methods use the same controller, types of runtime state information, and execution budget. Each game is evaluated in three runs, starting from its initial state each time. For

Method	Token	Runtime	PHY	PLA	VIS
GameHatch3D	73.3K	9.8	74.8	75.5	73.1
w/o ES	90.5K	11.3	62.3	68.1	48.4
w/o CS	87.4K	10.5	64.0	69.4	58.3
w/o PS	76.4K	10.2	50.5	70.3	71.9
w/o RS	77.0K	10.1	71.7	62.2	72.3
w/o Skills	193.1K	12.5	45.2	58.6	47.5
w/o AF	54.2K	6.4	73.1	74.2	69.4
w/o GF	70.2K	8.2	70.4	71.2	72.3
w/o AF & GF	50.5K	5.4	68.4	69.2	68.8

Table 2: Ablation study of the construction skills and refinement feedback in GameHatch3D. ES, CS, PS, and RS denote Environment, Character, Physics, and Rule Skills. "AF" and "GF" denote aesthetic feedback and gameplay feedback, respectively.

Round	Token	Runtime	PHY	PLA	VIS
0	50.5K	5.4	68.4	69.2	68.8
1	61.3K	6.9	72.1	71.5	71.2
2	70.8K	8.3	73.5	73.7	72.2
3	73.3K	9.8	74.8	75.5	73.1
4	75.2K	11.3	75.2	76.2	73.6

Table 3: Ablation study on automated playtesting rounds. Round 0 denotes generation without playtesting-based refinement. "PHY", "PLA", and "VIS" denote physical correctness, playability, and visual quality, respectively.

Method	Token	Runtime	PHY	PLA	VIS
HatchCraft	50.5K	5.4	68.4	69.2	68.8
w/o CCU	42.4K	4.6	52.1	64.3	68.2

Table 4: Ablation of cross-component updates (CCU), with identical skills and assembler. Both variants are evaluated before playtesting-based refinement.

Method	Gameplay Experience	Visual Quality
Cursor	35%	15%
GameHatch3D	65%	85%

Table 5: User preferences between games generated by Cursor and GameHatch3D, evaluated by 20 participants.

each run, we calculate the percentage of intermediate subgoals completed and assign 100 for final-objective success or 0 otherwise. The run score is the average of these two values, and the final score is averaged across the three runs. This metric measures objective completion under the evaluation controller. (3) **Visual Quality.** We sample 48 frames across the three runs using the game's native camera and score them with the aesthetic-quality evaluator from VBench (Huang et al., 2024). The overall score is the arithmetic mean of physical correctness, playability, and visual quality.

4.2 Comparison with General-Purpose Agentic Baseline

We compare GameHatch3D with Cursor, using Claude Opus 4.8 and GPT-5.5 as backend models. We also evaluate the backend models as our baselines. Each setting is allowed multiple model calls until an executable game is produced, and we report the average token consumption and end-to-end runtime. For GameHatch3D, the reported runtime includes three rounds of automated playtesting and refinement. As shown in Tab. 1, Cursor improves game quality over direct backend generation through its built-in planning, implementation, and debugging workflow, but incurs substantially higher token consumption and longer runtime. GameHatch3D achieves a better trade-off between generation quality and token consumption. Its four reusable skills provide structured guidance for core game components, allowing the backend model to focus on parameterization and task-specific adaptation. This design enables GameHatch3D to produce games with more accurate physics, richer visual details, and better playability while consuming fewer tokens than Cursor. It also maintains an end-to-end runtime comparable to that of Cursor while performing three rounds of automated playtesting. In practice, Cursor may require manual intervention and additional refinement calls to correct incomplete or erroneous outputs, further increasing model usage and human effort.

Fig. 5 further presents qualitative comparisons between Cursor and GameHatch3D. Cursor captures the coarse scene layout but produces relatively simplified environments and less expressive gameplay interactions. In contrast, GameHatch3D better realizes the intended visual composition and interaction structure. For the temple duel, it generates a more detailed combat environment with clearer character actions and visual feedback. For the squad battle, it better preserves the multi-character setup and turn-based confrontation while providing a richer and more organized arena. These results further demonstrate the benefits of the structured game-construction skills and auto-

mated refinement process for improving visual completeness and playability. **The supplementary material** provides gameplay video demos and a library of generated games spanning multiple genres, and demonstrates that modifying the input prompts can produce variants with different visual styles within the same genre.

4.3 Ablation Study

We conduct ablation studies to assess the contribution of each component in GameHatch3D, as shown in Tab. 2. The skill-level ablations demonstrate that the four skills play complementary roles. Among the individual skill ablations, removing the Environment Skill causes the largest degradation in visual quality, while removing the Physics Skill most strongly affects physical correctness. The Rule Skill primarily contributes to playability, whereas the Character Skill benefits all three quality dimensions. Removing any individual skill increases token consumption, with the removal of the Environment or Character Skill resulting in particularly pronounced increases. Removing all four skills deprives the agent of domain-specific guidance, leading to substantial degradation in physical correctness, playability, and visual quality, while also increasing generation cost. Ablating either refinement signal in automated playtesting degrades the corresponding quality dimensions. The efficiency results further show that visual refinement accounts for most of the additional overhead, as scene-level revisions often require extensive regeneration, particularly for large open-world exploration maps. In contrast, gameplay refinement introduces comparatively modest overhead because interactions and rules can be adjusted through high-level parameterized interfaces over registered game entities, avoiding extensive code regeneration. We further vary the number of playtesting rounds, as reported in Tab. 3. Additional rounds generally improve generation quality but also increase runtime. We therefore use three rounds by default, as this setting provides a favorable trade-off between generation quality and execution efficiency. We further evaluate CCU in Tab. 4. Removing CCU limits the framework's ability to resolve inconsistencies among interdependent components, leading to reduced game quality. These results highlight the importance of coordinating dependencies among game components, whose complementary roles jointly support the construction of coherent and playable games. To complement the automated evaluation, we conduct a user study with 20 participants comparing games generated by GameHatch3D and Cursor. As reported in Tab. 5, 65% of participants prefer GameHatch3D for gameplay experience, while 85% prefer it for visual quality. These results provide additional evidence of the benefits of GameHatch3D from the players' perspective, with a stronger preference for its visual quality.

5 Conclusion

In this paper, we present GameHatch3D, an end-to-end agentic framework for generating 3D games. We revisit the foundational components of game creation and distill four reusable domain-specific skills covering scene construction, character creation, physical behavior modeling, and gameplay logic implementation. These skills are coordinated by HatchCraft, our game-construction harness, which manages their execution, information exchange, and cross-component updates to produce coherent and executable games. To bridge the gap between executability and playability, we introduce an engine-guided playtesting mechanism with dual-path self-correction. Engine-provided state information supports navigation planning and action generation, while playtesting observations and failures enable the diagnosis and correction of visual and gameplay defects. We further establish a benchmark comprising diverse gameplay demonstrations and curated high-level game descriptions, together with a comprehensive evaluation protocol. Experimental results show that GameHatch3D generates higher-quality and more playable games than a general-purpose agentic baseline while consuming fewer tokens.

6 Limitations

Our work focuses on enabling general-purpose agents to generate playable 3D games with reduced token consumption. The current implementation uses Three.js for rendering and relies primarily on programmatically constructed assets, leaving room for richer geometry and visual detail. Future work could incorporate higher-quality 3D assets to improve visual fidelity and extend the framework to Unity or Unreal Engine.

AI USE STATEMENT

In this work, we used Claude Opus 4.8 and GPT-5.5 as backend models for game generation, automated playtesting, and iterative refinement. We also used these models to generate and cross-check high-level game specifications from gameplay videos, and used Claude Opus 4.8 to assist visual assessment of physical correctness. Human reviewers verified and corrected flagged specification details against the source videos. Generated games were evaluated through execution-based tests and automated playtesting, as described in Section 4.1. Additionally, we used generative AI tools to assist with manuscript drafting and language editing. The authors reviewed the AI-assisted manuscript text and take responsibility for the final content of this work, including its text, claims, experimental results, and generated artifacts.

ETHICS STATEMENT

GameHatch3D aims to support research on automated game creation. Our benchmark draws on gameplay videos from Gaming-500-Hours and YouTube. These source materials and their associated game content remain subject to their respective licenses and rights. Generated games should undergo content and security review before public deployment.

REPRODUCIBILITY STATEMENT

Section 3 describes the GameHatch3D framework, including the four domain-specific skills and engine-guided automated playtesting and refinement. Section 4.1 details benchmark construction, specification verification, and the evaluation metrics and protocol. Sections 4.2 and 4.3 describe the baseline comparisons and ablation studies, respectively. The supplementary material provides gameplay video demonstrations. We will open-source the agent framework and domain-specific skills to facilitate reproducibility and support further research on automated game generation within the community.

REFERENCES

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent S: An open agentic framework that uses computers like a human. *CoRR*, abs/2410.08164, 2024.

Anthropic. Claude opus 4.8 system card. https://www.anthropic.com/claude-opus-4-8-system-card, 2026.

Anysphere. Cursor documentation. https://cursor.com/docs, 2026. Accessed: 2026-07-28.

Adrian Bolton, Alexander Lerchner, Alexandra Cordell, Alexandre Moufarek, Andrew Bolt, Andrew Lampinen, Anna Mitenkova, Arne Olav Hallingstad, Bojan Vujatovic, Bonnie Li, et al. Sima 2: A generalist embodied agent for virtual worlds. *CoRR*, abs/2512.04797, 2025.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. In *ICLR*, 2025.

Haoxuan Che, Xuanhua He, Quande Liu, Cheng Jin, and Hao Chen. Gamegen-x: Interactive open-world game video generation. In *ICLR*, 2025.

Sam Earle, Samyak Parajuli, and Andrzej Banburski-Fahey. Demonstrating dreamgarden: A designer assistant for growing games from a single prompt. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, 2025.

Epic Games. Unreal engine. https://www.unrealengine.com, 1998.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, et al. Vbench: Comprehensive benchmark suite for video generative models. In *CVPR*, 2024.

Yilei Jiang, Jinyuan Hu, Qianyin Xiao, Yaozhi Zheng, Ruize Ma, Kaituo Feng, Jiaming Han, Tianshuo Peng, Kaixuan Fan, Manyuan Zhang, et al. Opengame: Open agentic coding for games. *CoRR*, abs/2604.18394, 2026.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *ICLR*, 2024.

Seth Karten, Jake Grigsby, Tersoo Upaa Jr, Junik Bae, Seonghun Hong, Hyunyoung Jeong, Jaeyoon Jung, Kun Kerdthaisong, Gyungbo Kim, Hyeokgi Kim, et al. The pokeagent challenge: Competitive and long-context learning at scale. *CoRR*, abs/2603.15563, 2026.

Jiaqi Li, Junshu Tang, Zhiyong Xu, Longhuang Wu, Yuan Zhou, Shuai Shao, Tianbao Yu, Zhiguo Cao, and Qinglin Lu. Hunyuan-gamecraft: High-dynamic interactive game video generation with hybrid history condition. *CoRR*, abs/2506.17201, 2025.

Loïc Magne, Anas Awadalla, Guanzhi Wang, Yinzhen Xu, Joshua Belofsky, Fengyuan Hu, Joohwan Kim, Ludwig Schmidt, Georgia Gkioxari, Jan Kautz, et al. Nitrogen: An open foundation model for generalist gaming agents. In *CVPR*, 2026.

Markov AI. Gaming-500-Hours. `https://huggingface.co/datasets/markov-ai/gaming-500-hours`, 2026. Dataset, accessed July 29, 2026.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *ACM Transactions on Intelligent Systems and Technology*, 16(5):1–72, 2025.

OpenAI. Gpt-5.5 system card. `https://openai.com/index/gpt-5-5-system-card/`, 2026.

Thang D Pham, Aditya Tanikanti, and Murat Keçeli. Chemgraph as an agentic framework for computational chemistry workflows. *Communications Chemistry*, 2026.

Mike Roberts, Renhan Wang, Rushikesh Zawar, Rachith Dey-Prakash, Quentin Leboutet, Stephan R Richter, Matthias Müller, German Ros, Rui Tang, Stefan Leutenegger, et al. Spear: A simulator for photorealistic embodied ai research. *CoRR*, abs/2607.06701, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *NeurIPS*, 2023.

Unity Technologies. Unity game engine. `https://unity.com`, 2005.

Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. Diffusion models are real-time game engines. In *ICLR*, 2025.

Hui Wu, Chao Xu, Jianghui Wang, Ziqiong Liu, Dong Li, Yiwei Dai, and Emad Barsoum. Moec: A memory-routed mixture-of-experts controller for adaptive minecraft control. In *ACL*, 2026.

Hongchi Xia, Xuan Li, Zhaoshuo Li, Qianli Ma, Jiashu Xu, Ming-Yu Liu, Yin Cui, Tsung-Yi Lin, Wei-Chiu Ma, Shenlong Wang, et al. Sage: Scalable agentic 3d scene generation for embodied ai. In *CVPR*, 2026.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *NeurIPS*, 2024a.

Runxin Yang, Yuxuan Wan, Shuqing Li, and Michael R Lyu. 90% faster, 100% code-free: Mllm-driven zero-code 3d game development. *CoRR*, abs/2509.26161, 2025.

Yandan Yang, Baoxiong Jia, Peiyuan Zhi, and Siyuan Huang. Physcene: Physically interactable 3d scene synthesis for embodied ai. In *CVPR*, 2024b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *11th International Conference on Learning Representations, ICLR 2023*, 2023.

Lei Yin, Wentao Cheng, Zhida Qin, Tianyu Huang, Yidong Li, and Gangyi Ding. Autoue: Automated generation of 3d games in unreal engine via multi-agent systems. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026.

Yuguang Yue, Irakli Salia, Samuel Hunt, Christopher Green, Wenzhe Shi, and Jonathan J Hunt. Learning to play: A multimodal agent for 3d game-play. *CoRR*, abs/2510.16774, 2025.

Jingyi Zhang, Jiaxing Huang, Sheng Jin, and Shijian Lu. Vision-language models for vision tasks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 46(8):5625–5644, 2024.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *CoRR*, abs/2303.18223, 2023.

Junhao Zheng, Chengming Shi, Xidi Cai, Qiuke Li, Duzhen Zhang, Chenxing Li, Dong Yu, and Qianli Ma. Lifelong learning of large language model based agents: A roadmap. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026.

A Appendix

A.1 Implementation Details

Skill contents. Each skill is described in Markdown and specifies the construction tasks it covers, required inputs, and output interfaces. It also provides construction instructions, code examples, and notes on common errors. During game generation, the backend model receives instructions tailored to the game's requirements and the skill guidance to generate game-specific code. For example, the Environment Skill defines the construction order for terrain, lighting, and scene objects, provides a code skeleton, and requires the visible terrain and collision handling to use the same ground-height function to prevent characters from floating or sinking.

Development process. We use Claude Code with Claude Opus 4.8 to generate an initial collection of 100 games covering seven gameplay categories: navigation, platforming, shooting, combat, collection, puzzles, and racing. We manually debug and refine the generated Three.js implementations to improve playability and visual quality. From these refined implementations, the model summarizes recurring construction procedures and drafts reusable code examples. We review the summaries and examples against the implementations and execution records, correct errors, and organize the resulting guidance into the four skills. Although the library is developed from seven gameplay categories, we subsequently evaluate it on a benchmark spanning 21 game genres. Its application across this broader benchmark suggests that the extracted procedures capture shared requirements for scene construction, character creation, physical interactions, and gameplay rules, supporting reuse across different game genres.

Separation and versioning. We prepare the development specifications independently of the evaluation videos and check the development and evaluation specifications for duplicate scenarios and near-duplicate prompts. Basic gameplay mechanics may be shared across the two sets. We fix the skill library before generating games for evaluation and record the repository commit and skill-file hashes for each run. Any subsequent change to the library creates a new version and requires reevaluating all compared conditions.

Execution and assembly. HatchCraft constructs Environment, Character, Physics, and Rule in sequence. Each step receives the specification and earlier component descriptions. It outputs a JSON record containing entity identifiers, references, configurable properties, and code for game-specific behavior. Proposed changes to earlier records are validated and made available to subsequent steps. The assembler creates scene objects, attaches the generated scripts, and connects events through the shared runtime.

A.2 Comparison with Codex

We further compare GameHatch3D with Codex using GPT-5.5 as the shared backend. As shown in Table 6, Codex achieves slightly higher playability than Cursor but lower physical correctness

Backend	Method	Token (K)↓	Runtime (min)↓	Physics↑	Playability↑	Visual Quality↑	Total Score↑
GPT-5.5	Direct	12.1	3.8	23.7	27.1	32.3	27.7
	Cursor	204.8	10.9	50.4	60.5	52.9	54.6
	Codex	220.3	11.8	48.4	62.3	50.2	53.6
	GameHatch3D	65.9	9.2	72.6	73.5	73.2	73.1

Table 6: Comparison of direct backend generation, Cursor, Codex, and GameHatch3D using GPT-5.5 as the shared backend.

and visual quality, with higher token consumption and longer average runtime. GameHatch3D outperforms both agents across all three quality dimensions. Compared with Codex, it improves the overall score from 53.6 to 73.1 while reducing token consumption and runtime, demonstrating its effectiveness and efficiency against an additional general-purpose agentic baseline.