GAMECRAFTER: AN AGENTIC FRAMEWORK FOR MULTI-GAMEPLAY GAME WORLD GENERATION

Anonymous authors
Paper under double-blind review



Figure 1: **3D game worlds generated by GameCrafter.** The generated environments support (a) exploration and boss combat gameplay modes and (b) first-person, third-person, and 2.5D camera views. (c) Aligned RGB videos, skeleton, whitebox, depth, and semantic segmentation data are captured from the same simulation state and camera configuration.

ABSTRACT

Recent advances in generative models and agentic workflows are making it increasingly feasible to automate game creation. These workflows offer a path from high-level user intent to playable worlds. However, existing systems largely focus on isolated assets, scene construction, or games centered on a single gameplay pattern, leaving coherent multi-gameplay 3D world generation underexplored. Generating such worlds requires coordinating scene construction, character creation, gameplay implementation and different softwares. To this end, we present GameCrafter, an agentic framework for scalable multi-gameplay game world generation. The long and interdependent production process makes consistent coordination across specialized agents and tools particularly challenging; GameCrafter addresses this with **Specification-Guided Orchestration** under shared production requirements. Moreover, visual plausibility alone does not guarantee gameplay readiness, motivating **Structure-Aware Asset Synthesis** that jointly models appearance and executable structure for environments and characters. These assets are then grounded in a shared executable runtime through **Multi-Gameplay World Realization**, allowing them to coexist consistently within the same world. Since errors can accumulate across stages and only become apparent after runtime integration, GameCrafter further employs **Closed-Loop Quality Control**, combining visual inspection, playtesting, and iterative repair. Beyond playable worlds, GameCrafter also produces synchronized multi-view, multimodal data aligned within the same simulation. Experiments demonstrate strong performance in visual quality, gameplay fidelity, and overall playability compared with existing game-generation approaches. Project Page: https://anonymous.4open.science/w/GameCrafter_27805

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

1 INTRODUCTION

Recent advances in generative models and autonomous agents are rapidly lowering the barrier to game content creation, making it increasingly feasible to construct interactive virtual worlds from high-level user intent Earle et al. (2025); Bruce et al. (2024); Yin et al. (2026); Guo et al. (2026). Beyond reducing the cost of game prototyping and development, scalable world generation also holds promise for applications such as embodied simulation, interactive content creation, and large-scale data generation Pfaff et al. (2026); Li et al. (2026); Xu et al. (2026). However, a complete game world requires more than visually plausible assets or scenes: it must support coherent interaction and diverse gameplay within a unified, executable environment.

Recent efforts toward automatic game creation can be broadly grouped into three directions. Visual world generation methods focus on synthesizing high-quality scenes, assets, or explorable 3D environments from text or visual inputs, but typically emphasize appearance and geometry rather than complete gameplay functionality Sun et al. (2023); Bruce et al. (2024); Valevski et al. (2025); Che et al. (2025); Yu et al. (2025); Guo et al. (2026); Huang et al. (2026b). Code-centric and agentic game generation approaches instead generate executable game logic, yet are often developed for relatively constrained settings such as 2D or browser-based games Earle et al. (2025); Jiang et al. (2026); Huang et al. (2026a). More recently, engine-integrated systems have begun to connect foundation models with professional game engines and content-creation software, enabling increasingly complete pipelines that combine asset preparation, scene construction, gameplay implementation, and automated testing Tang et al. (2025); Yin et al. (2026). Despite this progress, existing systems largely target a single game specification or interaction pattern, leaving the generation of high-fidelity 3D worlds that coherently support multiple gameplay modes within the same environment underexplored Earle et al. (2025); Jiang et al. (2026); Yin et al. (2026); Guo et al. (2026).

Realizing such coherent multi-gameplay worlds is non-trivial, as it requires resolving several coupled challenges across the entire game-production pipeline. First, constructing a complete 3D game world requires coordinating heterogeneous agents and software across scene design, asset creation, gameplay logic, rendering, and testing, where errors or inconsistencies introduced at one stage can easily propagate downstream Yin et al. (2026); Jiang et al. (2026). Second, visually plausible content is not necessarily gameplay-ready: environments must satisfy spatial constraints, while characters must support rigging, animation, collision, and interaction Pfaff et al. (2026); Wang et al. (2026); Tang et al. (2025). Third, supporting multiple gameplay modes further requires these components to remain consistent within a shared runtime, while many failures can only be exposed through actual execution and playtesting Huang et al. (2026a); Yin et al. (2026). These challenges suggest that scalable multi-gameplay generation should be formulated not as one-shot content synthesis, but as a structured, executable, and verifiable world-building process.

To address these challenges, we introduce GameCrafter, an agentic framework that generates a coherent multi-gameplay 3D game world from a single prompt. GameCrafter organizes specialized agents and heterogeneous software tools around a shared structured specification, progressively turning high-level intent into gameplay-ready assets and integrating them into a common executable runtime where multiple gameplay modes coexist consistently. By coupling structured coordination with runtime validation and iterative repair, GameCrafter transforms game generation from a collection of loosely connected production steps into a unified and verifiable world-building process.

Specifically, GameCrafter follows a four-stage pipeline that progressively transforms a high-level prompt into a verified executable world. Stage 1, **Specification-Guided Orchestration**, converts user intent into structured production requirements and coordinates specialized agents and heterogeneous software through shared intermediate artifacts. Building on this specification, Stage 2, **Structure-Aware Asset Synthesis**, constructs gameplay-ready environments, players, and bosses by jointly modeling visual appearance and executable structure, including spatial layouts, exploration routes, geometry, rigs, and animations. These assets are then assembled in Stage 3, **Multi-Gameplay World Realization**, into a shared runtime where exploration, combat, and multiple camera perspectives operate consistently within the same world. Finally, Stage 4, **Closed-Loop Quality Control**, combines visual assessment with runtime validation to detect failures, localize their causes, and iteratively repair the corresponding production stages. The resulting executable worlds further enable synchronized multi-view and multimodal data collection within the same simulation. Exten-

sive experiments demonstrate strong performance in visual quality, gameplay fidelity, and overall playability compared with existing game-generation systems.

To sum up, our contributions are as follows:

- **Multi-gameplay game world generation.** We introduce GameCrafter, to our knowledge the first agentic framework for generating coherent 3D game worlds that support multiple gameplay modes within a shared executable environment. Unlike systems that generate isolated scenes or single-purpose game experiences, GameCrafter enables exploration and combat to coexist in the same world while supporting multiple camera perspectives.
- **An agentic workflow for end-to-end world creation.** We develop a framework that coordinates specialized agents and heterogeneous software across asset creation, scene construction, character rigging and animation, game-engine integration, and quality control. Through structured intermediate artifacts and closed-loop validation, the workflow turns loosely coupled generation tools into a unified and recoverable world-building process.
- **Synchronized multimodal game-world data.** GameCrafter produces aligned multi-view and multimodal observations directly from the generated executable worlds, including rendered RGB, structural whitebox, depth maps, and semantic representations. This provides a scalable source of paired interactive data while preserving consistency across gameplay modes, camera viewpoints, and data modalities.

2 Related Work

2.1 Playable 3D Game and World Generation

Recent work has expanded language-guided content creation from individual assets to structured environments and playable games. UnrealLLM connects language agents with Unreal Engine's procedural content generation system to construct controllable and interactable scenes (Tang et al., 2025). WorldClaw combines hierarchical planning, terrain construction, asset generation, and render-based refinement to produce large-scale worlds with editable components (Guo et al., 2026). Game-development agents additionally coordinate simple scene construction with executable gameplay. DreamGarden uses hierarchical planning and iterative execution to develop game prototypes (Earle et al., 2025). AutoUE integrates asset retrieval, scene generation, gameplay and interaction code synthesis within Unreal Engine (Yin et al., 2026). OpenGame targets web games through specialized coding models, reusable development skills, and execution-based evaluation (Jiang et al., 2026).

These systems therefore either emphasize world construction or generate executable games around a relatively fixed interaction pattern. In contrast, GameCrafter targets coherent 3D worlds in which multiple gameplay modes coexist within the same executable environment.

2.2 Agentic Workflows for Multi-Tool 3D Production

Language agents increasingly mediate access to specialized content creation software. 3D-GPT decomposes modeling tasks among specialized agents and translates language instructions into procedural operations compatible with Blender (Sun et al., 2023). 3Dify combines retrieval-augmented generation, tool protocols, and GUI interaction to automate digital content creation tools (Hayashi et al., 2025). Beyond tool invocation, recent systems incorporate structured coordination and feedback. SceneSmith organizes simulation-ready scene construction through hierarchical interactions among designer, critic, and orchestrator agents (Pfaff et al., 2026). SceneCode represents scenes through executable asset programs and a persistent state registry (Wang et al., 2026).

Existing agentic systems demonstrate effective tool use and execution-based refinement, but typically optimize individual content creation tasks or bounded development workflows. GameCrafter instead coordinates heterogeneous agents and software tools across the full game-production process under shared specifications and artifact dependencies.

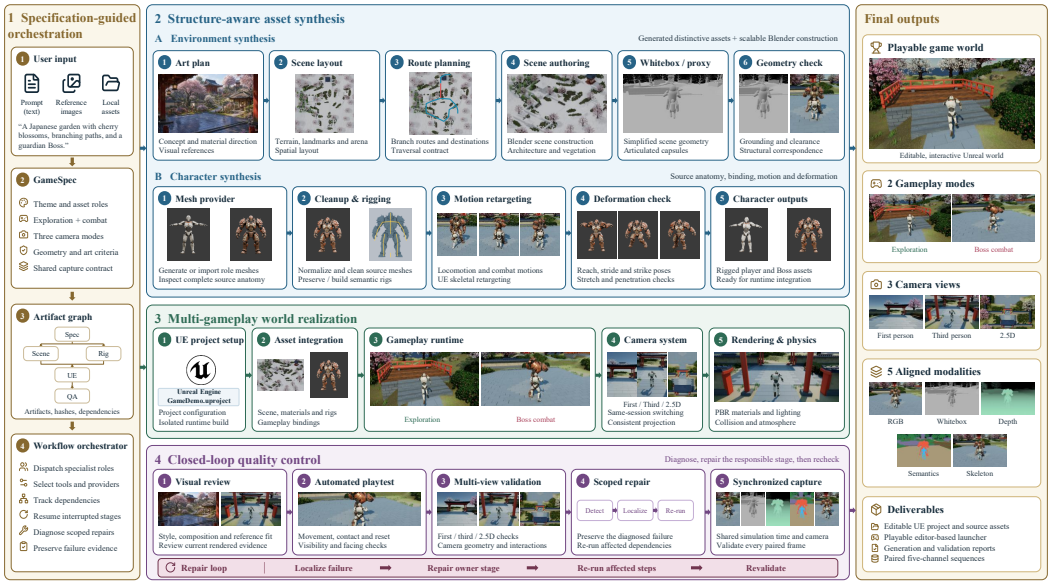


Figure 2: **Overview of the GameCrafter framework.** (1) Specification-guided orchestration converts user inputs into a shared game specification and organizes production through an artifact dependency graph. (2) Structure-aware asset synthesis creates gameplay-ready environments and characters. (3) Multi-gameplay world realization integrates these assets into a shared Unreal Engine. (4) Closed-loop quality control combines visual inspection and automated playtesting to localize failures, repair responsible stages, and re-execute affected downstream steps. The framework produces editable, playable worlds and synchronized multimodal observations.

2.3 Data Generation for Interactive World Models

Training world models requires data that capture how environments evolve under actions, while controllable rendering benefits from geometric and appearance information. Pairing visual observations with actions, scene structure, and temporal correspondences provides supervision beyond unannotated video. WildWorld collects action-conditioned gameplay from a commercial action role-playing game, with synchronized annotations of character skeletons, world states, camera poses, and depth (Li et al., 2026). Generative World Renderer pairs RGB video with geometry and material G-buffers for inverse rendering and controllable visual synthesis (Huang et al., 2026b). Magpie provides temporally aligned high-fidelity and white-box videos in Unreal Engine (Zhan et al., 2026).

Existing data engines provide structured observations through gameplay recording and configurable rendering. However, scaling these pipelines to diverse environments and interaction settings, or adapting their outputs to different supervision requirements, often requires additional engineering effort. GameCrafter instead integrates world and gameplay authoring with synchronized multimodal capture, enabling variation in source environments, interaction rules, camera viewpoints, and visual configurations within a unified production pipeline.

3 Method

Given a high-level user request u, which may contain text prompts, reference images, and user-provided assets, our goal is to construct a complete executable game world $\mathcal{W}$ that supports multiple gameplay modes and multiple camera perspectives, while simultaneously producing synchronized multimodal observations. As shown in Figure 2, our method consists of the following four stages: (1) Specification-Guided Orchestration, (2) Structure-Aware Asset Synthesis, (3) Multi-Gameplay World Realization, and (4) Closed-Loop Quality Control.

3.1 SPECIFICATION-GUIDED ORCHESTRATION

Shared game specification. Constructing a complete game world requires coordination across heterogeneous tasks, including environment design, character creation, animation, scene assembly, gameplay programming, rendering, and validation. Allowing individual agents to perform these tasks independently can easily lead to inconsistent assumptions about scale, layout, asset interfaces, or gameplay requirements. We therefore introduce a **specification-guided orchestration** mechanism that provides a shared contract throughout the entire creation process. Given user input u, a design agent first converts the high-level request into a structured game specification

$$\mathcal{S} = f_{\text{design}}(u), \tag{1}$$

where $\mathcal{S}$ describes the intended visual style, environment structure, characters, gameplay requirements, camera configurations, and acceptance criteria. Importantly, the specification defines not only *what* should be generated, but also the interfaces between different production stages.

Artifact-centric orchestration. GameCrafter represents the creation process as an artifact dependency graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node $v_i \in \mathcal{V}$ corresponds to a production artifact, such as a scene layout, character mesh, rigged model, animation set, or executable level, and each edge represents a dependency between artifacts. Specialized agents operate on these artifacts rather than communicating solely through unconstrained natural-language messages. For an agent $\mathcal{A}_i$, we write

$$r_i = \mathcal{A}_i(\mathcal{S}, \mathcal{R}_i), \tag{2}$$

where $\mathcal{R}_i$ denotes the set of prerequisite artifacts and r_i is the newly generated or modified artifact.

A central orchestrator resolves dependencies, selects generation or editing tools, dispatches tasks to specialist agents, and tracks results. This design provides a consistent interface across heterogeneous software and enables partial execution: completed artifacts can be reused, while downstream stages affected by a modification can be selectively recomputed. As a result, GameCrafter turns a loosely coupled collection of agents and tools into a structured and recoverable world-building process.

3.2 STRUCTURE-AWARE ASSET SYNTHESIS

A visually plausible asset is not necessarily usable in an interactive game. An environment may appear realistic while containing disconnected paths or invalid scene geometry, while a visually appealing character may still fail under rigging, animation, or collision. We therefore formulate asset generation as **structure-aware synthesis**, in which each generated asset contains both a visual representation and an executable structural representation:

$$x = \left(x^{\text{app}}, x^{\text{str}}\right), \tag{3}$$

where x^{app} captures appearance and x^{str} captures the structural properties required for gameplay.

Gameplay-ready environments. For environment generation, GameCrafter first derives an art plan from the specification, including visual references, material families, and reusable scene assets. In parallel, a layout agent constructs the metric structure of the world, including terrain organization, landmarks, boss arenas, and spatial connectivity. An exploration planner further determines traversable routes, branching paths, destinations, and transitions between gameplay regions. These constraints guide scene construction in a 3D content-creation environment, where terrain, roads, architecture, and decorative assets are assembled into a coherent scene.

In addition to the rendered environment, we construct a structural proxy representation that preserves gameplay-relevant geometry while removing unnecessary visual detail. This representation captures navigable surfaces, openings, collision-relevant objects, and spatial boundaries, providing a bridge between visual scene synthesis and downstream runtime integration.

Animation-ready characters. Player and boss meshes are generated or retrieved independently, followed by geometry normalization, mesh cleanup, skeleton construction or adaptation, skinning, and animation retargeting. We validate each character under a set of representative poses and motions to identify common deformation failures such as incorrect joint behavior, mesh adhesion, or missing skinning weights. The resulting character package therefore contains not only appearance assets, but also the kinematic structure required for runtime animation and interaction.

3.3 MULTI-GAMEPLAY WORLD REALIZATION

The synthesized assets must ultimately be grounded into a common executable environment. We therefore introduce **multi-gameplay world realization**, which integrates heterogeneous assets into a shared runtime rather than creating independent scenes for individual gameplay demonstrations.

Unified runtime. We represent an executable world as

$$\mathcal{W} = (\mathcal{G}_{\text{scene}}, \mathcal{C}, \mathcal{P}, \mathcal{V}), \tag{4}$$

where $\mathcal{G}_{\text{scene}}$ denotes scene geometry and environment assets, $\mathcal{C}$ denotes characters and animations, $\mathcal{P}$ denotes gameplay and interaction logic, and $\mathcal{V}$ denotes camera and rendering configurations.

In our implementation, these components are assembled inside a commercial game engine. Scene geometry, materials, collision, characters, skeletons, and animations are first imported into a unified world coordinate system. Runtime logic then instantiates movement, interaction, combat, damage, death, reset, and other relevant gameplay behaviors. Environment rendering further integrates lighting, physically based materials, atmospheric effects, and engine-level rendering features to preserve the original visual quality of the generated 3D assets.

Multiple gameplay modes. A key property of GameCrafter is that different gameplay modes operate over the *same* persistent world. Let $m \in \mathcal{M}$ denote a gameplay mode, such as exploration or boss combat. Rather than constructing a separate world $\mathcal{W}_m$ for each mode, GameCrafter executes different modes over a shared $\mathcal{W}$. Consequently, geometry, characters, spatial coordinates, physics, and world state remain consistent when transitioning between exploration and combat.

Multiple camera views. The same world can also be observed through multiple camera configurations $v \in \mathcal{V}_{\text{cam}}$, including first-person, third-person, and orthographic or 2.5D viewpoints. This shared-runtime formulation enables diverse gameplay and observation settings without duplicating or independently regenerating the underlying world.

3.4 CLOSED-LOOP QUALITY CONTROL

Game-world generation contains failure modes that cannot be detected reliably from static assets alone. For example, a character may appear correct in a neutral pose but deform incorrectly during combat, while a scene may look visually complete yet contain blocked paths, invalid collision, or inconsistent camera behavior. GameCrafter therefore incorporates a **closed-loop quality control** process that validates both visual quality and executable behavior.

Visual and runtime validation. We collect two complementary forms of evidence, $\mathcal{E} = \mathcal{E}_{\text{vis}} \cup \mathcal{E}_{\text{run}}$, where $\mathcal{E}_{\text{vis}}$ contains visual evidence such as rendered views, character poses, and appearance comparisons, while $\mathcal{E}_{\text{run}}$ contains evidence obtained from actual execution, including automated playtests, camera checks, geometric validation, and gameplay-state measurements.

Visual review agents evaluate properties that are difficult to express through deterministic rules, such as style consistency, scene composition, character appearance, and visible deformation artifacts. Executable checks complement them by testing properties such as route connectivity, collision validity, animation continuity, combat interaction, death and reset behavior, and camera consistency. A world is accepted only when all required quality gates are satisfied.

Targeted repair. When validation fails, GameCrafter associates the failure with the production artifact and agent responsible for that property. Rather than restarting the pipeline, the system repairs the corresponding stage, invalidates only dependent downstream artifacts, and resumes execution:

$$e_{\text{fail}} \rightarrow v_i \rightarrow \text{Repair}(v_i) \rightarrow \text{Recompute}(\text{Desc}(v_i)). \tag{5}$$

Synchronized data capture. Once a world passes validation, the same runtime is also used for data capture. For a world state $\mathcal{W}_t$ and camera v, GameCrafter records aligned observations

$$\mathbf{o}_t^v = \left\{ I^{\text{RGB}}, I^{\text{whitebox}}, I^{\text{depth}}, I^{\text{sem}}, \ldots \right\}, \tag{6}$$

where all modalities share the same simulation state and camera configuration. By repeating this process across gameplay modes and viewpoints, GameCrafter naturally produces a multi-view, multimodal dataset while preserving alignment across gameplay, geometry, and visual observations.

4 EXPERIMENTS

We evaluate **GameCrafter** from two complementary perspectives, reflecting its dual role in both playable game creation and synchronized multimodal data production:

- **Q1: Scene Quality.** How well does GameCrafter generate visually coherent, spatially structured, and prompt-aligned 3D scenes and characters for use as both playable game content and sources of aligned multimodal data?
- **Q2: Playability.** Can GameCrafter integrate the generated content into playable games that satisfy the specified gameplay requirements, execute mechanics correctly, and provide clear feedback during both exploration and boss combat?

4.1 EXPERIMENTAL SETUP

4.1.1 BENCHMARK

We construct a benchmark of ten tasks, each combining a shared gameplay specification with a scene-specific description. All tasks require a controllable humanoid player, a humanoid melee boss, connected exploration and combat spaces. The scene descriptions vary architectural styles, route topology, elevation, materials, illumination, and character appearance. The tasks are grouped into two easy, three medium, three hard, and two very hard cases according to design complexity.

4.1.2 SETTINGS

Baselines. We compare GameCrafter with three baselines. AutoUE (Yin et al., 2026) automates Unreal Engine game development through asset retrieval, scene generation, gameplay code synthesis, and automated playtesting. Codex with 3AGameFactory (OpenDCAI, 2026) augments the coding agent with specialized game-development tools and workflows; we use its UE5 workflow for scene assembly and gameplay implementation. Standalone Codex uses its general-purpose coding and tool-use capabilities to build games without a dedicated game-production workflow.

Evaluation Protocol and Implementation. We adopt a VLM-based evaluation protocol to assess scene quality and playability. All baselines and GameCrafter use gpt-6-astra with high reasoning effort for generation. Evaluation is also conducted by Codex using the same model and reasoning configuration. For each generated game, the evaluator receives the task specification and a fixed evaluation prompt containing the playtesting procedure and scoring rubrics. Scores are grounded in visual observations and interaction outcomes, with supporting screenshots and brief explanations. All methods follow the same evaluation procedure and playtesting budget. Each evaluation takes place in a fresh session without access to generation history or previous scores, and evaluation feedback is not returned to the generation process.

Metrics. We adapt the four-dimensional evaluation framework of GameCraft-Bench (Luo et al., 2026) to our 3D benchmark. *Core Mechanics* (M) evaluates whether the requested gameplay loop functions correctly, including player movement, melee combat, dodging, hit reactions, and encounter reset. *Content Depth* (D) measures the realization of the requested playable content, including distinct areas, route structures, exploration coverage, and scene elements. *Functional Visuals* (V) assesses the readability of gameplay information, including action feedback, character visibility, and navigation cues. *Art and Presentation* (A) evaluates the visual coherence and quality of both characters and environments, including architectural style, materials, lighting, and character design.

We evaluate each game using fixed, task-specific rubrics grounded in observed gameplay evidence. The rubric comprises 18 items: six for M and four each for D, V, and A. Each item is further decomposed into atomic checks, with task-specific requirements determining the applicable content and spatial checks. M/D/V checks use scores of 0, 0.5, and 1 for no, partial, and clear demonstrated fulfillment, respectively. A checks use quality ratings from 0 to 4, normalized to $[0, 1]$. The five levels correspond to clear failure (0), rough (1), competent (2), excellent (3), and outstanding (4). Unobserved checks contribute zero and remain in the denominator. We average check scores within each item, then item scores within each dimension, and rescale to 0–100. The overall score is

$$\text{Overall} = 0.15\,\text{M} + 0.35\,\text{D} + 0.15\,\text{V} + 0.35\,\text{A}. \tag{7}$$

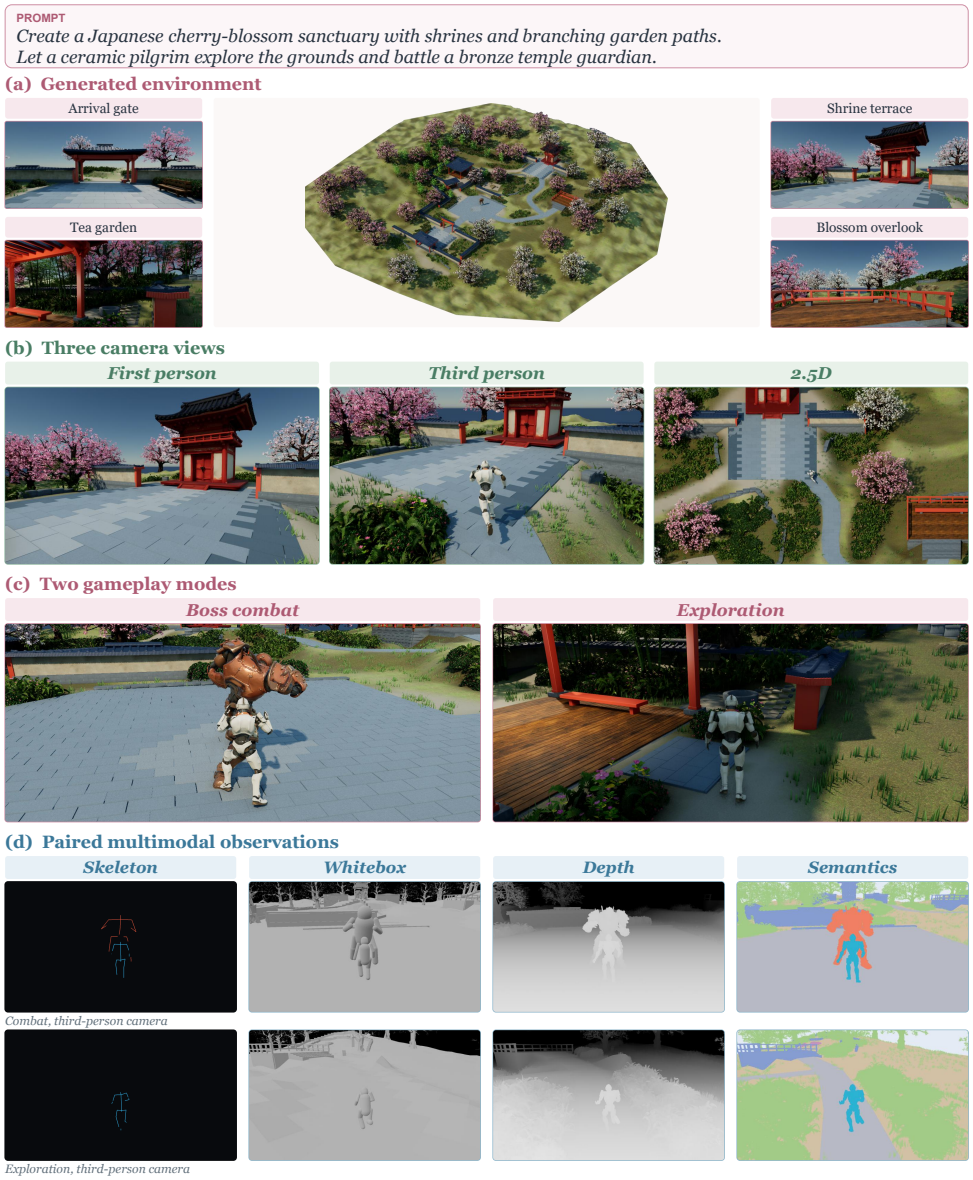


Figure 3: **A Japanese cherry-blossom sanctuary generated by GameCrafter from the prompt shown above.** (a) Environment overview and representative locations. (b) First-person, third-person, and 2.5D camera views. (c) Boss combat and exploration within the generated world. (d) Paired skeleton, whitebox, depth, and semantic observations for combat (top row) and exploration (bottom row), captured from the corresponding third-person views.

We first average scores across runs within each task and then equally across the ten tasks.

4.2 Qualitative Results

As shown in Figure 4, we compare GameCrafter with the baseline methods on benchmark tasks 05 and 07. GameCrafter produces detailed, visually coherent environments alongside high-quality character models and animations. In comparison, 3AGameFactory and Codex produce coarse, low-polygon environments with limited detail, while AutoUE uses detailed assets but arranges them with poor spatial coherence. Gameplay observations further reveal character deformation and animation artifacts in 3AGameFactory, whereas Codex and AutoUE exhibit overly simplistic character models and animations, limiting their overall visual polish.

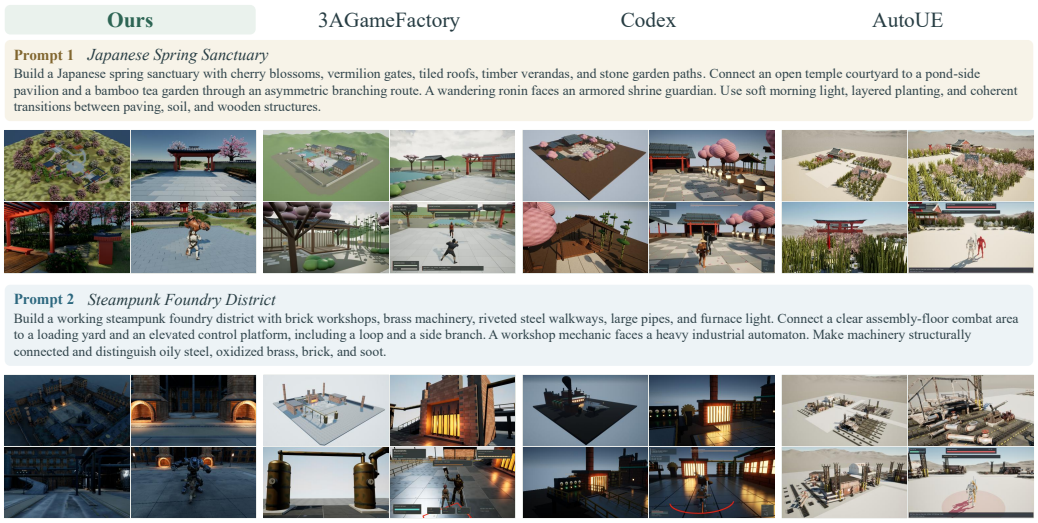


Figure 4: **Qualitative comparison on two benchmark tasks.** GameCrafter is compared with 3AGameFactory, Codex, and AutoUE using the same prompt for each task: *Japanese Spring Sanctuary* (top) and *Steampunk Foundry District* (bottom).

Method	M ↑	D ↑	V ↑	A ↑	Overall ↑
AutoUE	79.58	37.48	55.21	43.75	48.65
Codex	98.33	92.03	83.75	49.22	76.75
Codex + 3AGameFactory	**100**	92.51	95.21	52.81	80.14
Codex + GameCrafter(Ours)	**100**	**93.37**	**96.86**	**73.04**	**87.77**

Table 1: Quantitative comparison on the ten-task 3D game-generation benchmark. All scores are reported on a 0–100 scale; higher is better for all metrics.

4.3 Quantitative Results

Table 1 shows that Codex + GameCrafter achieves the highest overall score of **87.77** on the ten-task benchmark, improving upon standalone Codex by **11.02** points and Codex + 3AGameFactory by **7.63** points. Compared with Codex + 3AGameFactory, both methods reach **100.00** in Core Mechanics, while GameCrafter yields modest gains in Content Depth (**+0.86**) and Functional Visuals (**+1.65**). The largest difference is in Art and Presentation, where the score increases from **52.81** to **73.04**. Under the benchmark's weighting, this dimension accounts for approximately **7.08** points of the overall improvement. These results suggest that GameCrafter's main contribution is stronger visual presentation and quality, accompanied by comparable gameplay functionality and slightly better content coverage and gameplay readability.

5 Conclusion

We presented GameCrafter, an agentic framework that combines structured production and execution-guided repair to generate playable 3D worlds. It supports exploration, boss combat, multiple camera views, and synchronized multimodal capture. Experiments show that GameCrafter outperforms existing game-generation baselines in overall performance. Our current evaluation focuses on exploration and humanoid melee combat. Future work includes extending the framework to more diverse gameplay mechanics, improving generation and repair efficiency, and evaluating the utility of the generated multimodal data for downstream world-model training.

AI USE STATEMENT

In this work, we used generative AI tools to polish the writing and to assist with parts of the code implementation. Generative models are also components of GameCrafter itself: as described in Section 3, the framework uses LLM-based agents to coordinate production and uses image and 3D generation models to create reference images and 3D assets, and the multimodal observations are rendered from the resulting worlds. Our evaluation protocol (Section 4) likewise uses an LLM-based evaluator to score the generated games. Apart from these uses, which are part of the method and experimental design described in this paper, we did not use generative AI tools to generate or alter experimental results, analyze data, or formulate the main scientific claims. All AI-assisted text was reviewed and revised by the authors, and AI-assisted code was manually inspected, tested, and validated for correctness. We take responsibility for the final content of this work, including all text, claims, code, and artifacts produced with the aid of generative AI.

REFERENCES

Jake Bruce, Michael Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Bechtle, Feryal Behbahani, Stephanie Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando de Freitas, Satinder Singh, and Tim Rocktäschel. Genie: Generative Interactive Environments. *arXiv preprint arXiv:2402.15391*, 2024. URL https://arxiv.org/abs/2402.15391.

Haoxuan Che, Xuanhua He, Quande Liu, Cheng Jin, and Hao Chen. GameGen-X: Interactive Open-World Game Video Generation. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8VG8tpPZhe.

Sam Earle, Samyak Parajuli, and Andrzej Banburski-Fahey. DreamGarden: A Designer Assistant for Growing Games from a Single Prompt. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025. doi: 10.1145/3706598.3714233. URL https://doi.org/10.1145/3706598.3714233.

Chunchao Guo, Jinpeng Li, Yang Li, and Zilong Huang. WorldClaw: Agentic 3D Open-World Generation at Scale. *arXiv preprint arXiv:2608.05248*, 2026. URL https://arxiv.org/abs/2608.05248.

Shun-ichiro Hayashi, Daichi Mukunoki, Tetsuya Hoshino, Satoshi Ohshima, and Takahiro Katagiri. 3Dify: a Framework for Procedural 3D-CG Generation Assisted by LLMs Using MCP and RAG. *arXiv preprint arXiv:2510.04536*, 2025. URL https://arxiv.org/abs/2510.04536.

Yixu Huang, Bo Li, Na Li, Zhe Wang, Kaijie Chen, Haonan Ge, Qingyi Si, Yuanzhe Shen, Ruihan Yang, Guangjing Wang, and Hongcheng Guo. GUI Agents for Continual Game Generation. *arXiv preprint arXiv:2605.28258*, 2026a. URL https://arxiv.org/abs/2605.28258.

Zheng-Hui Huang, Zhixiang Wang, Jiaming Tan, Ruihan Yu, Yidan Zhang, Bo Zheng, Yu-Lun Liu, Yung-Yu Chuang, and Kaipeng Zhang. Generative World Renderer. *arXiv preprint arXiv:2604.02329*, 2026b. URL https://arxiv.org/abs/2604.02329.

Yilei Jiang, Jinyuan Hu, Qianyin Xiao, Yaozhi Zheng, Ruize Ma, Kaituo Feng, Jiaming Han, Tianshuo Peng, Kaixuan Fan, Manyuan Zhang, and Xiangyu Yue. OpenGame: Open Agentic Coding for Games. *arXiv preprint arXiv:2604.18394*, 2026. URL https://arxiv.org/abs/2604.18394.

Zhen Li, Zian Meng, Shuwei Shi, Wenshuo Peng, Yuwei Wu, Bo Zheng, Chuanhao Li, and Kaipeng Zhang. WildWorld: A Large-Scale Dataset for Dynamic World Modeling with Actions and Explicit State toward Generative ARPG. *arXiv preprint arXiv:2603.23497*, 2026. URL https://arxiv.org/abs/2603.23497.

Tongxu Luo, Rongsheng Wang, Jiaxi Bi, Chenming Xu, Zhengyang Tang, et al. GameCraft-Bench: Can agents build playable games end-to-end in a real game engine? *arXiv preprint arXiv:2606.17861*, 2026.

OpenDCAI. 3AGameFactory: Open-Source 3A Game Generation Skills and Asset Framework. Open-source software repository, 2026. URL https://github.com/OpenDCAI/GameFactory-3A.

Nicholas Pfaff, Thomas Cohn, Sergey Zakharov, Rick Cory, and Russ Tedrake. SceneSmith: Agentic Generation of Simulation-Ready Indoor Scenes. *arXiv preprint arXiv:2602.09153*, 2026. URL https://arxiv.org/abs/2602.09153.

Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 3D-GPT: Procedural 3D Modeling with Large Language Models. *arXiv preprint arXiv:2310.12945*, 2023. URL https://arxiv.org/abs/2310.12945.

Song Tang, Kaiyong Zhao, Lei Wang, Yuliang Li, Xuebo Liu, Junyi Zou, Qiang Wang, and Xiaowen Chu. UnrealLLM: Towards highly controllable and interactable 3D scene generation by LLM-powered procedural content generation. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19417–19435, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.994. URL https://aclanthology.org/2025.findings-acl.994/.

Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. Diffusion Models Are Real-Time Game Engines. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=P8pqeEkn1H.

Puyi Wang, Yuhao Wang, Linjie Li, Zhengyuan Yang, Kevin Qinghong Lin, Yangguang Li, and Yu Cheng. SceneCode: Executable World Programs for Editable Indoor Scenes with Articulated Objects. *arXiv preprint arXiv:2605.19587*, 2026. URL https://arxiv.org/abs/2605.19587.

Xiaojie Xu, Zhengyuan Lin, Runyi Li, Yihao Liu, Kaipeng Zhang, and Yongtao Ge. WorldRover: A Scalable Synthetic Video Data Engine for World Exploration with Rich Annotations. *arXiv preprint arXiv:2608.15659*, 2026. URL https://arxiv.org/abs/2608.15659.

Lei Yin, Wentao Cheng, Zhida Qin, Tianyu Huang, Yidong Li, and Gangyi Ding. AutoUE: Automated Generation of 3D Games in Unreal Engine via Multi-Agent Systems. *arXiv preprint arXiv:2603.07106*, 2026. URL https://arxiv.org/abs/2603.07106.

Jiwen Yu, Yiran Qin, Xintao Wang, Pengfei Wan, Di Zhang, and Xihui Liu. GameFactory: Creating New Games with Generative Interactive Videos. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11590–11599, 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Yu_GameFactory_Creating_New_Games_with_Generative_Interactive_Videos_ICCV_2025_paper.html.

Xiaoyu Zhan, Xinyu Wang, Xiaohong Zhang, Huanjie Zhu, Tengjiao Sun, Pengcheng Fang, Jiaxing Yu, Yanwen Guo, and Dongjie Fu. Magpie: Real-Time World Renderer for Interactive Games. *arXiv preprint arXiv:2608.27168*, 2026. URL https://arxiv.org/abs/2608.27168.

A End-to-End World Generation

GameCrafter turns a prompt and optional reference images into an editable Unreal Engine (UE) world with exploration, Boss combat, three camera modes, and paired observations. Design, production, and review share versioned specifications, assets, an executable project, and evidence. Each handoff commits an artifact; each repair revises its responsible source and rechecks affected consumers.

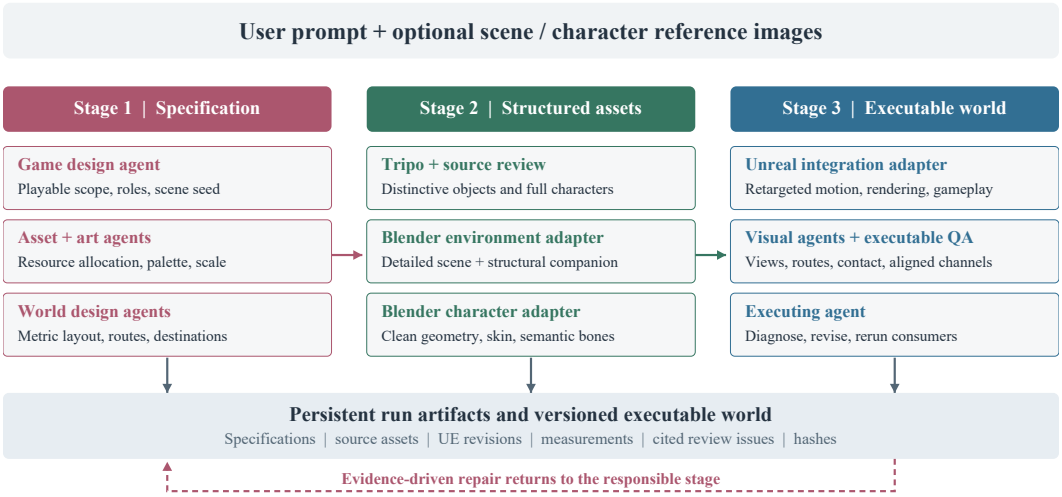


Figure 5: Artifact-mediated coordination. Roles share specifications and source-linked evidence. Arrows denote handoffs and repair dependencies; they do not imply simultaneous agent execution. Stages 1–3 in the figure correspond to Sections 3.1–3.3; the review and repair paths implement Closed-Loop Quality Control (Section 3.4).

Roles and tools. Model roles plan or review; deterministic adapters validate and execute; the coding agent diagnoses and repairs. Tripo supplies source geometry, Blender prepares assets and constructs the scene, and UE realizes motion, rendering, collision, gameplay, and capture. Art and layout planning consume resources that actually arrived, so specification and production interleave.

Execution scope. The executor is sequential, while independent remote environment requests may overlap. The sections below follow specification, structured construction, and executable-world validation within its finer dependency graph. Case-specific inputs and settings are stated in each case figure.

A.1 Stage 1: Specification-Guided Orchestration

Table 2: Planning roles and their committed decisions.

Role	Input	Output
Game design	Brief, references, supported schema	World identity, character roles, gameplay, concept prompt
Asset planning	Design, concept, resource menu, budget	Generated objects versus authored construction; scale and source criteria
Art direction	Concept, prepared sources, observed albedo	Resource-grounded palette, material roles, regional identity
World layout	Design, art plan, geometry grammar	Metric terrain, buildings, encounter core, spawns
Exploration planning	Current scene, destinations, constraints	Branches, widths, elevations, itinerary and return

A.1.1 Visual References and Production Specifications

The design agent specifies architecture, palette, lighting, an encounter space, and traversal destinations. Its concept prompt goes to `gpt-image-2`: generation for text input, editing for a scene reference. The image guides appearance and spatial organization; metric layout is committed separately. The complete prompts are provided in the supplementary material.

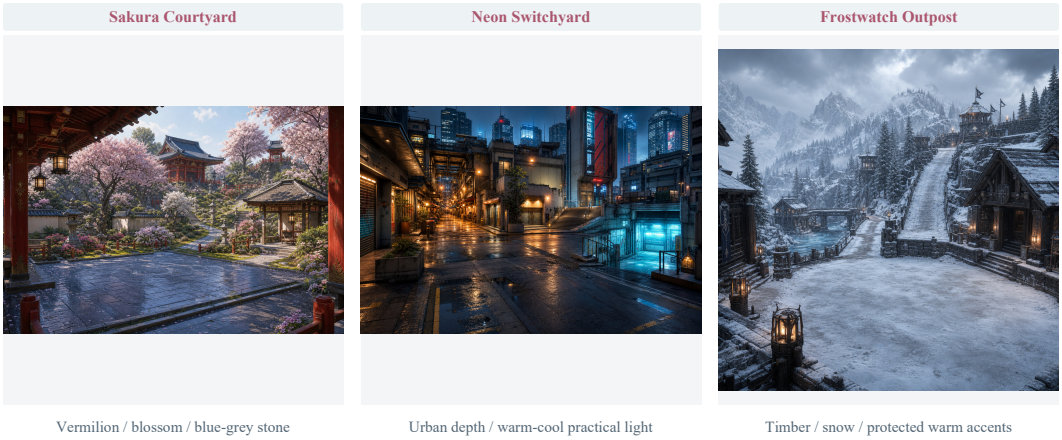


Figure 6: Actual concept outputs. Sakura, Neon, and Frostwatch references establish distinct architectural, material, and spatial targets for downstream production.

A.2 Stage 2: Structure-Aware Asset Synthesis

A.2.1 Environment Assets and Material Preparation

Tripo generates distinctive buildings, plants, and equipment from text or images. Blender authors scalable and traversable structures. The adapter persists task IDs, polls, downloads, then prepares bounds, scale, UVs, materials, and structural proxies. Source review checks completeness, useful gaps, and unwanted bases; matched PBR maps retain provenance. Generated buildings serve as exterior landmarks unless access is explicitly authored and tested.

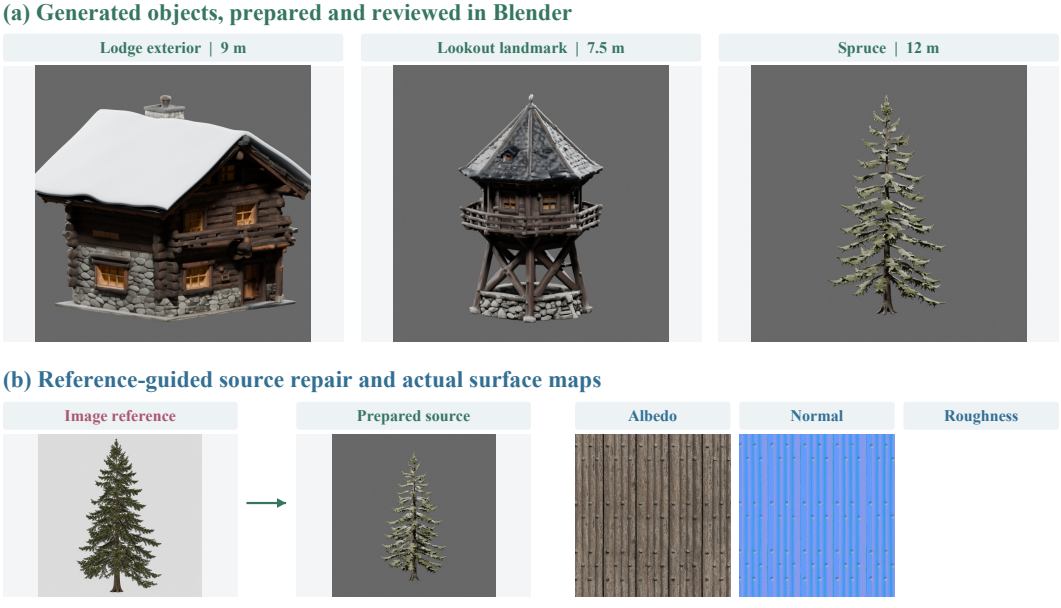


Figure 7: Frostwatch source assets. Prepared lodge, lookout, and spruce; an image-guided spruce revision; and matched wood PBR maps. Blender inspection renders are distinct from final UE imagery.

A.2.2 Map Layout, Exploration Routes, and Structural Whitebox

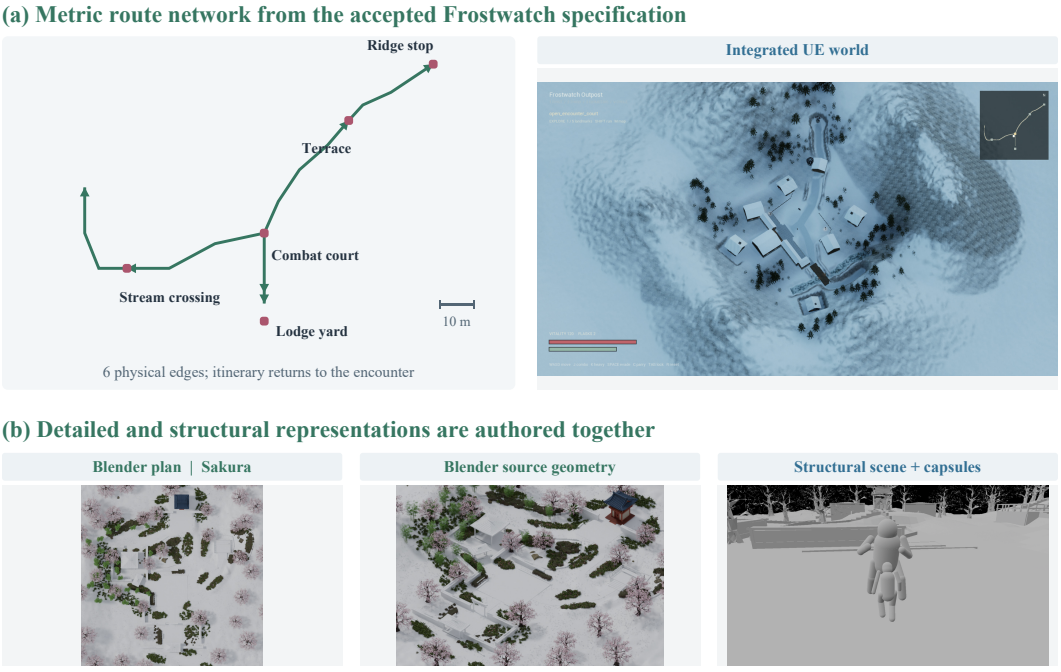


Figure 8: Spatial design and geometric realization. Frostwatch's authored route coordinates and UE overview; Sakura's Blender plan/oblique views and native whitebox. The route diagram is a specification, not a player trace.

A metric, playable layout. Routes carry control points, widths, elevations, surface ownership, and an ordered itinerary. Frostwatch contains six physical edges connecting five destinations: encounter court, lodge workyard, stream crossing, observation terrace, and ridge signal stop. The itinerary visits branches and returns to combat. Destinations vary in function, enclosure, elevation, and silhouette; props and vegetation remain outside route clearances.

Table 3: Spatial contracts become construction and runtime checks.

Contract	Blender realization	Acceptance evidence
Terrain and routes	Fitted paving, foundations, ramps, stairs	Saved support geometry and collision-enabled UE traversal
Destinations	Distinct masses, openings, regional materials	Arrival views, clearance, route continuity
Source placement	Metric scale, transforms, planting regions	Bounds, corridor exclusion, camera visibility
Structural companion	Terrain, major masses, floors, openings	Shared transforms and camera; bounded geometric error

Whitebox is built alongside detail. Blender produces simplified terrain and architecture; generated sources contribute structural proxies. Articulated capsules follow evaluated character bones. These share camera and time with RGB, while being excluded from normal-play collision, shadows, and indirect lighting. The current terrain contract bounds its 95th-percentile height error to 18 cm; simplified silhouettes need not match detailed RGB boundaries.

A.2.3 Character Cleanup, Rigging, and Skinning

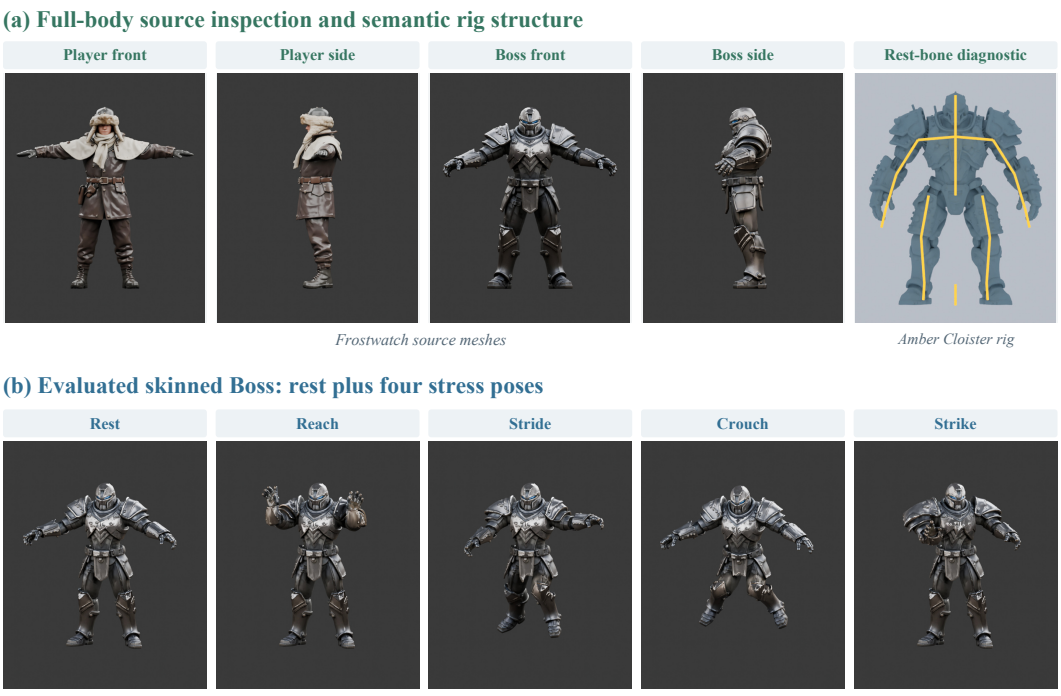


Figure 9: Source and skin checks. Frostwatch player/Boss inspections; Amber Cloister rest-bone diagnostic; and Frostwatch Boss deformations in rest, reach, stride, crouch, and strike poses.

Inspect, clean, bind. Character prompts request a complete unarmed figure with visible feet and separated limbs. Source review checks anatomy, landmarks, and forward direction. Blender conservatively welds near-coincident vertices and removes degeneracies or tiny flecks while preserving UV seams and anatomical gaps. Usable existing rigs are retained. Otherwise, source-aware landmarks place a humanoid armature; anatomical-chain weighting or bone heat creates real vertex weights and an armature modifier, with rigid-region overrides where needed.

Verify deformation. Rest plus four moving poses test edge stretch, spikes, displacement, and silhouette. Missing limbs or fused anatomy require source repair; bad weights require rebinding. UE rechecks deformation under retargeted motion.

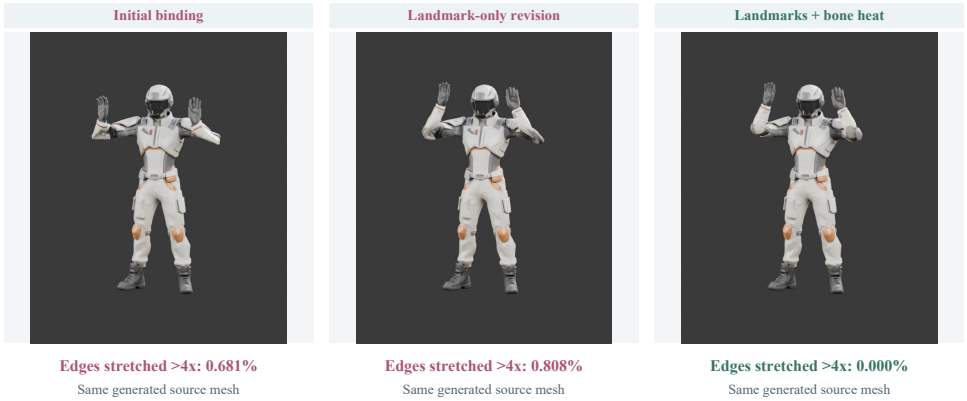


Figure 10: Neon skinning repair on one unchanged mesh. Reach-pose edges stretched beyond $4\times$: 0.6809% initially, 0.8082% after landmark-only repair, and 0% with source-aware landmarks plus bone heat. These measurements illustrate a single repair and are not a controlled ablation.

A.3 Stages 3–4: Multi-Gameplay World Realization and Closed-Loop Quality Control

A.3.1 UE Integration, Animation, and Gameplay

Assemble one shared runtime. UE imports scene geometry, materials, skins, and bone mappings; configures lighting and collision; and connects exploration and a single humanoid melee Boss encounter. Health, contact, hit reactions, death, reset, and physical return from exploration are checked in the same world. Delivery includes an editable project and editor launcher.

Retarget motion. Semantic IK chains map installed UE motions onto each character. The 14 required slots cover idle, four walk directions, jog, three light attacks, charged attack, dash, two hit reactions, and death. The implementation retargets this declared motion dependency rather than invoking a trained motion-generation service.

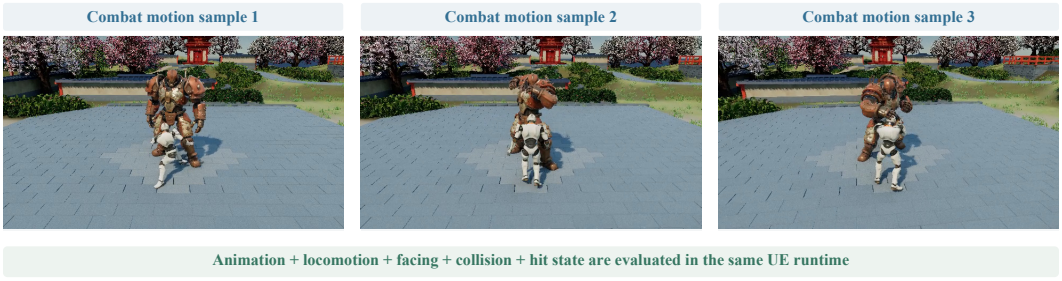


Figure 11: Integrated animation. Recorded Sakura combat poses show full-body motion; contact, health changes, and timing are verified separately in the runtime.

Table 4: Runtime acceptance across gameplay and cameras.

Condition	Required evidence
Exploration	Complete collision-enabled itinerary, destination coverage, physical Boss return
Combat	Consistent trajectories, attack contact, health/reactions, reset, visible hit frames
Facing	Travel and evaluated anatomy agree; a valid injected 90° error is rejected
Camera switching	Startup and first–third–isometric–third sequence; correct projection and restored visibility
Capture / performance	Active openings in all requested jobs; normal-play frame times measured separately

Camera switching and gameplay validation. First-person, third-person, and 2.5D modes switch within one session. The player body is hidden in first person and visible again in other modes. Neon's isometric dark bands were localized to short-range ambient occlusion; a projection-specific fix also had to run during startup, not only camera switching. This motivates checking initialization, transitions, and moving anatomy in the integrated project.

A.3.2 VISUAL REVIEW, TARGETED REPAIR, AND RECOVERY



Figure 12: Sakura RGB-transfer repair. Removing a redundant transfer reduced mean absolute 8-bit RGB error from 55.61 to 2.67 in a foreground stone region against the native reference, without changing scene lighting.

Review source-linked evidence. The reviewer compares current normal-rendered overviews, routes, and destinations with the brief, concept, and executable checks. Each issue records severity, cited evidence, defect, repair, and verification. Targeted geometry or rendering diagnostics distinguish ambiguous causes before reconstruction.

Table 5: Repair the layer responsible for the observed failure.

Failure	Owner	Recheck
Fused / missing anatomy	Source geometry	Completeness, deformation, retargeting
Webbing / stretched armor	Landmarks and skin weights	Stress poses, runtime motion
Blocked / floating route	Layout, support, collision	Saved geometry, complete traversal
Wrong-facing / hidden actor	Orientation and visibility	Moving anatomy, camera transitions
Washed-out / shifted output	Render target and export	Native buffers, independent calibration

Bounded iteration. The coding agent diagnoses, edits a new revision, and reruns affected consumers. Three distinct consecutive environmental refusals require a source-bound repair plan covering every issue before another preflight. Hard checks remain mandatory; optional polish stops after acceptance. This discipline does not guarantee success within four attempts. CLI auto-repair is limited to bounded exposure, sky, and fog patches; structural and gameplay changes remain agent work.

Recover without repeating valid work. Nodes commit named outputs and hashes; SQLite preserves state, remote task IDs, and failures. Configuration and source changes invalidate their dependents. Uncertain paid submissions are not silently repeated. Early capture covers every requested job; exports are reusable only when content, render policy, exporter, job settings, and output hashes match. Final validation still reads all frames. Fresh-source runs exclude previous-case meshes and levels.

A.3.3 Multimodal Capture and Alignment Verification

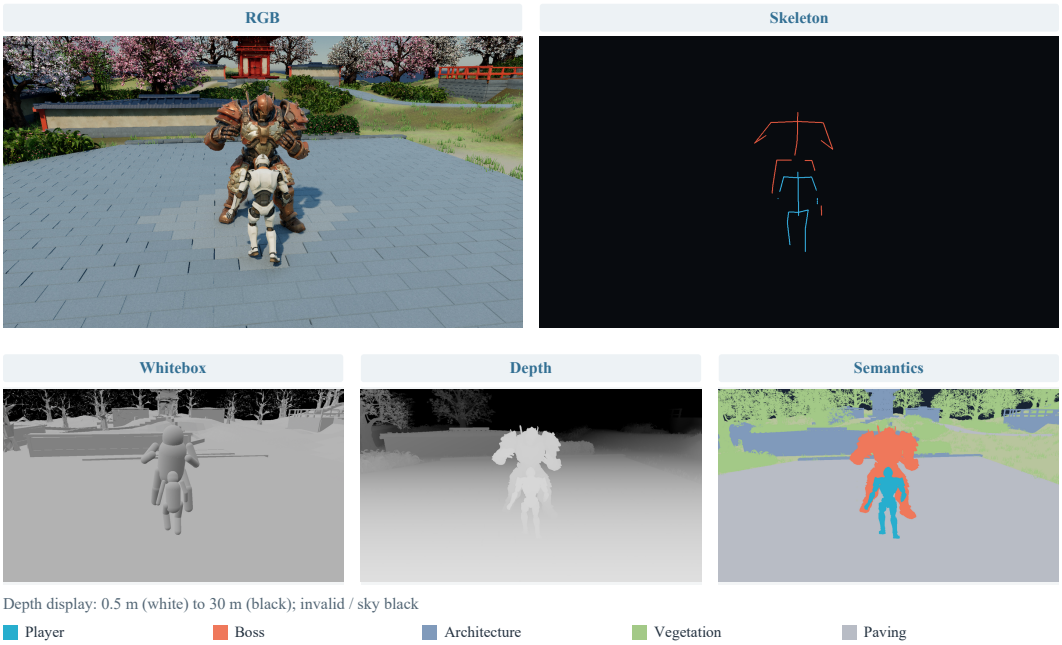


Figure 13: Five channels from one Sakura frame. Shared camera and simulation step align RGB, skeleton, whitebox, depth, and semantics. Depth/semantic colors are visualizations; simplified whitebox boundaries may differ from RGB.

Table 6: Authoritative observation formats.

Channel	Stored output	Interpretation
RGB	PNG / native color	HUD-free render with verified transfer
Depth	Float32 NPY	Camera-forward Z in metres
Semantics	UInt8 PNG	Fixed ontology; lossless label-video decoding
Skeleton	Joint / parent JSON	Post-animation joints, projections, visibility
Whitebox	PNG / structural buffer	Authored geometry and articulated capsules

Synchronization and verification. Capture follows animation and camera updates; frame manifests link channels and joints. Warmup is omitted, and temporal AA/TSR, motion blur, and depth of field are disabled. Known-distance geometry independently calibrates projection, depth, visibility, and structural alignment. Final checks compare every export with native buffers and verify skeleton/capsule timing and lossless semantics.

Scope. The default profile has three cameras $\times$ two activities, 360 frames each at 640×360 and 60 Hz simulation; this is not a real-time FPS claim. Each camera view is recorded in a separate run of the same world; within each view, all modalities are synchronized and pixel-aligned. Pairing applies within each camera/activity frame. Full-route testing is separate from these clips.

B End-to-End Case Studies

The following cases illustrate the complete workflow: world design, generated environments, three camera views, exploration and combat, and paired observations. Prompts are concise paraphrases; diagnostic examples are not controlled ablations. RGB panels may be fitted/cropped without aesthetic recoloring; modality rows retain complete common rasters. Depth display spans 0.5–30 m.

B.1 SAKURA COURTYARD

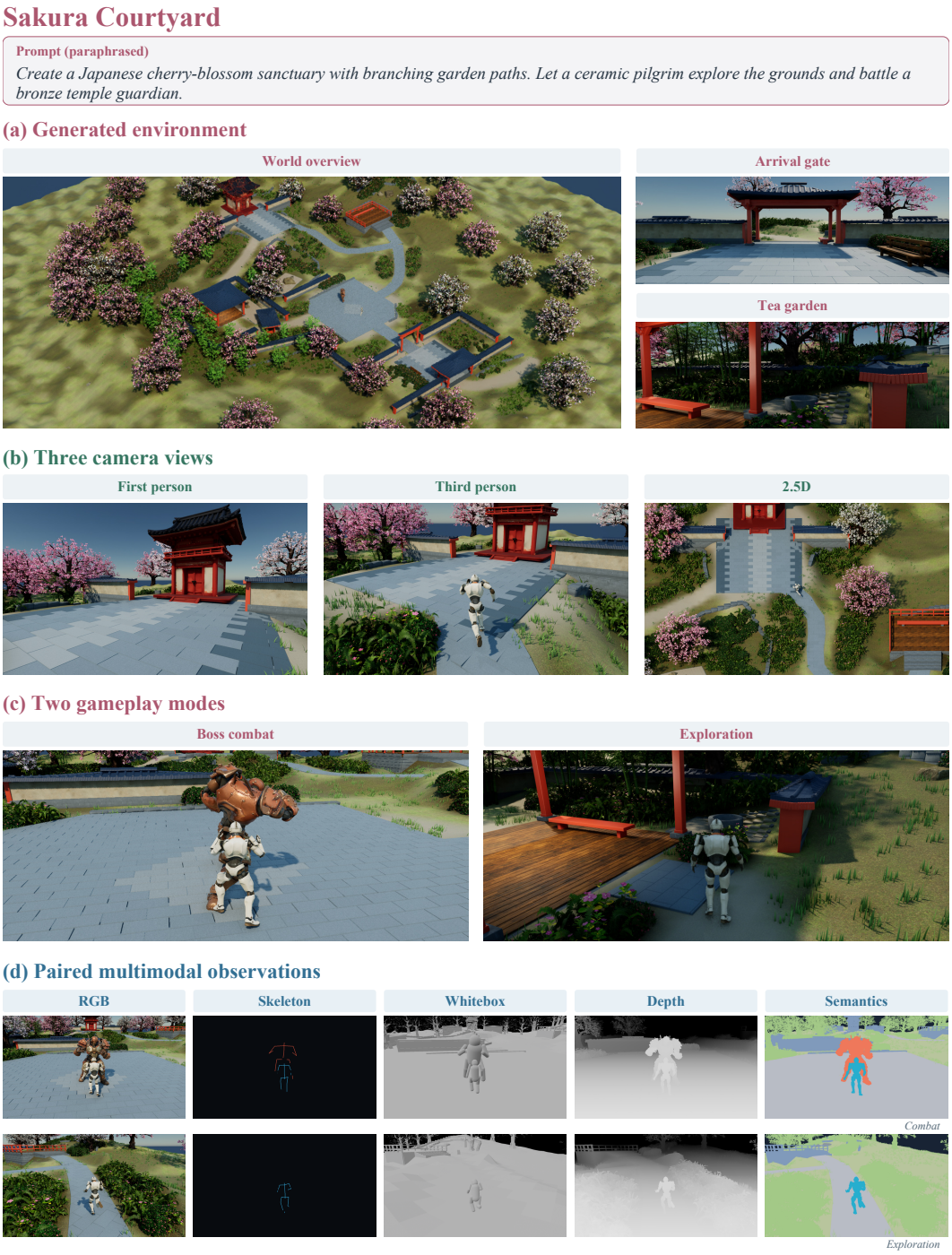


Figure 14: Sakura Courtyard. New Japanese environment with inherited character rigs and combat runtime. Each paired row uses frame 0 of a corrected native third-person session; combat and exploration are separate runs. The large combat panel is a separate action sample.

B.2 NEON SWITCHYARD

Neon Switchyard

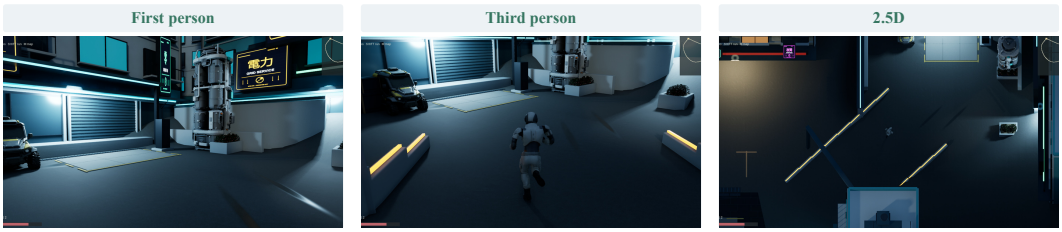
Prompt (paraphrased)

Create a neon-lit cyberpunk district with branching alleys and futuristic parked vehicles, where a courier explores and fights an armored enforcer.

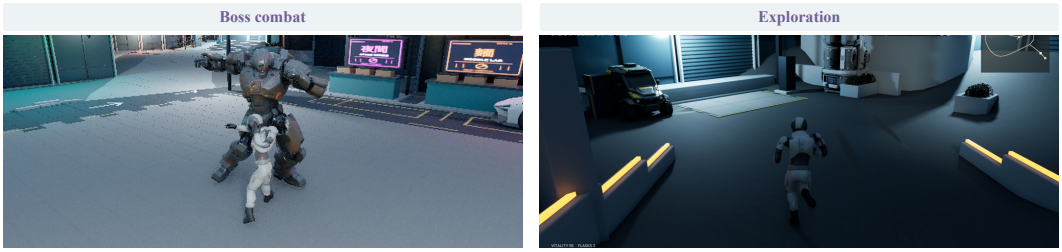
(a) Generated environment



(b) Three camera views



(c) Two gameplay modes



(d) Paired multimodal observations

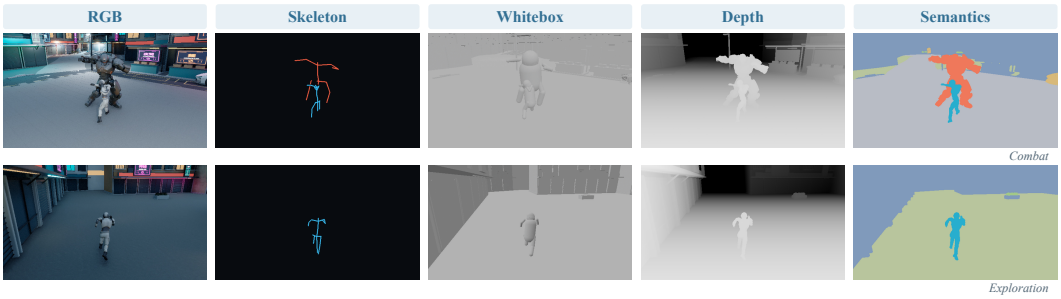


Figure 15: Neon Switchyard. Newly generated characters and distinctive urban sources; reviewed external resources were permitted by the case brief. Vehicles are props. Paired third-person rows use combat frame 120 and exploration frame 240; the three views show independently traversed repair-bay captures.

B.3 FROSTWATCH OUTPOST

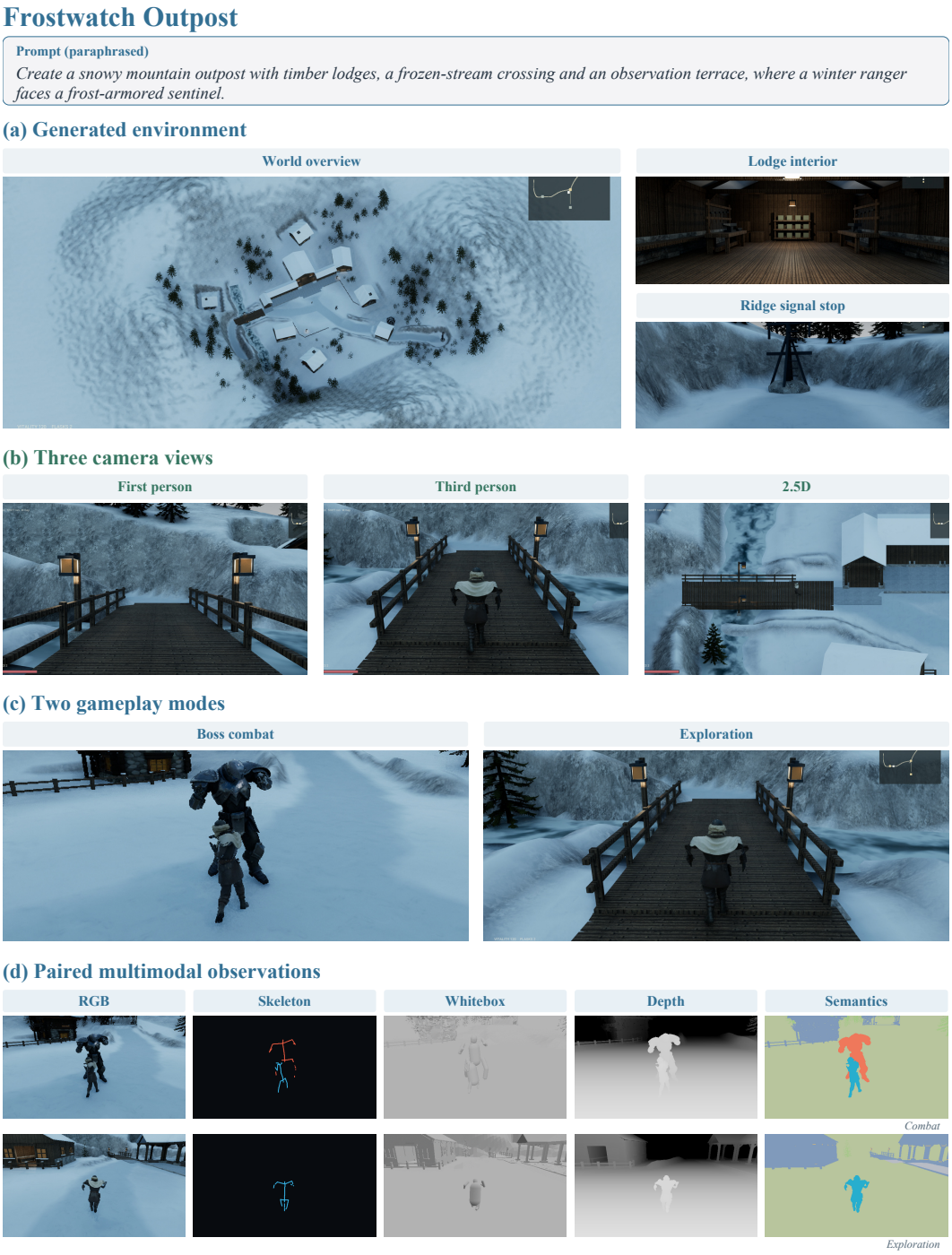


Figure 16: Frostwatch Outpost. Recorded fresh-source production, with public PBR and installed motions as dependencies. Paired third-person rows use combat frame 120 and exploration frame 240. Independent stream-crossing views show the connected winter world. These cases are qualitative illustrations and are not part of the benchmark in Section 4.

C Benchmark Evaluation Protocol

C.1 Rubric and Score Aggregation

The benchmark specification contains ten prompt-defined 3D games with a controllable humanoid, one melee Boss, and connected exploration and combat spaces. All cases share 18 items and fixed dimension weights; case-specific requirements yield 52–59 atomic checks. Table 7 separates functioning behavior, required content, readability, and visual quality.

Table 7: The four dimensions and all 18 scoring items. Item labels are shortened for presentation.

Dimension	Weight	Items
M: Core Mechanics	15%	M1 player control; M2 player melee; M3 melee Boss; M4 dodging; M5 hit reactions; M6 encounter reset.
D: Content Depth	35%	D1 area coverage; D2 route structure; D3 actual traversability; D4 required scene content.
V: Functional Visuals	15%	V1 attack telegraphs; V2 combat outcomes; V3 combat visibility and silhouettes; V4 navigation and terrain.
A: Art and Presentation	35%	A1 composition and landscape; A2 materials and finish; A3 lighting, color, and atmosphere; A4 characters, animation, and world coherence.

Two scoring scales. M/D/V checks receive 0, 0.5, or 1: not demonstrated/failed, partially supported, or fully supported. Evidence status distinguishes observed failure from missing coverage. A checks receive integer ratings $r \in \{0, 1, 2, 3, 4\}$: clear failure, rough, competent, excellent, or outstanding, normalized as $s = r/4$. Check-specific anchors refine these tiers. Missing art evidence has $r = \text{null}$ and $s = 0$, distinct from an observed quality failure. Limited evidence receives a scoped rating, never automatic half credit.

Table 8: The 16 equally weighted art checks, grouped by item.

Item	Atomic checks (in ID order, .1–.4)
A1	Visual focus/composition; spatial depth; architectural/landscape form; arrangement and density rhythm.
A2	Material quality; surface variation/repetition; seams/contact/grounding; object and architectural finish.
A3	Light shaping/value; palette and warm–cool relationships; time/weather/atmosphere; local light/shadow/reflection control.
A4	Character design; character finish; animation naturalness and dynamic deformation; character–world coherence.

Hierarchical aggregation. For item i with n_i checks and dimension d with item set I_d,

$$q_i = \frac{1}{n_i} \sum_{j=1}^{n_i} s_{ij}, \qquad C_d = \frac{100}{|I_d|} \sum_{i \in I_d} q_i, \tag{8}$$

$$O = 0.15C_M + 0.35C_D + 0.15C_V + 0.35C_A. \tag{9}$$

All checks remain in the denominator, including unobserved ones. This is not a flat mean of atomic checks. Each A item has four checks, so $C_A = 100 \sum r/64$, with null ratings contributing demonstrated zero. Evidence coverage is the fraction of checks with readable evidence and never rescales scores. Final 0–100 scores use two-decimal round-half-up, without intermediate rounding.

Relation to GameCraft-Bench. Our rubric keeps the M/D/V structure of GameCraft-Bench (Luo et al., 2026) and expands the art dimension into four items (A1–A4), including an explicit motion-quality check (A4.3). Because items and anchors differ, our scores are not directly comparable with those reported for GameCraft-Bench. The textual anchors of the A ratings have not been calibrated against human ratings.

C.2 Playtesting Procedure and Evaluator Prompts

The protocol separates actual playtesting from evidence-based judgment. Each game has the same six-minute maximum budget: two minutes each for combat/reset, exploration, and visual inspection. Generating-method identities and provisional difficulty labels are withheld from the judge. The blocks below condense the evaluation prompts; the full prompts and per-case checklists are provided in the supplementary material.

> **Playtesting prompt (condensed).** Using the supplied case-to-artifact mapping and controls, launch each game and verify its version. With actual player input, test movement, player/Boss attacks, anticipation, dodging an attack, both hit reactions, and reset followed by renewed combat. Traverse the specified routes as the player, including return paths, and inspect required areas, props, characters, surfaces, and lighting. Record screenshots/clips and input logs under the fixed budget. Do not modify the game. Free-camera flight, teleportation, and wall traversal do not demonstrate navigability. Distinguish game failures from evaluation-tool failures; report blocked tests rather than guessing.

> **Judge prompt (condensed; the full rubric remains authoritative).** Evaluate one artifact using the fixed version, `CASE_INPUT`, `RUN_REPORT`, `EVIDENCE_MANIFEST`, and readable visual evidence. Keep all checks and weights. Score each check once over the complete evidence package; include counterevidence rather than selecting the best demo.
>
> Use the M/D/V and A scales in Section C.1. Judge only requested features and observed behavior. Procedural assets, low polygon counts, minimalism, or nonphotorealistic styles are not deductions by themselves. Explain distinct consequences before applying one defect to multiple dimensions.
>
> Inspect representative wide, medium, and close player views. For A4.3, examine both characters in continuous sequences or sufficiently dense temporal samples: poses, weight, timing, transitions, and unintended stretching, reversed joints, collapse, or severe interpenetration. Record player/Boss observations separately, but issue one A4.3 rating. A still image cannot establish animation continuity or defect frequency.
>
> Return JSON with version and artifact identifiers; launch and scoring status; per-check score, evidence status, actual clips and timestamps or frame IDs, observation and reason; A raw rating, next-tier limitation, and scope limitations; item, dimension, and overall scores; evidence coverage; A evidence counts; missing evidence; and at most five principal issues. Never invent observations or timestamps.

Table 9: Execution failures and evidence gaps have different meanings.

Status	Treatment
`build_failed`	Confirmed inability to enter a renderable game scene: overall 0; dimension scores null; no invented check scores.
`insufficient_input`	Unknown launch status or wholly unreadable evidence: aggregates null; record missing inputs.
`scored_with_gaps`	Otherwise scoreable, but any check is unobserved or any A check has limited scope; retain all denominators.

Across cases. Average case totals equally, including confirmed build failures as zero. Missing-evidence cases remain unevaluated and prevent a complete ten-case claim. Report evaluated counts, failures, per-case coverage, and M separately. Dimension means use only artifacts with component observations and disclose their sample count. The protocol recommends a fixed judge version and human-reference calibration before formal comparison; report the judge configuration and calibration status alongside results.