nAIVE ENGINE

AI-Native Interactive Visual Engine

Product Requirements Document v2.0

"What if the AI doesn't just write the game — what if the AI IS the game?"

ASU Endless Games and Learning Lab

February 2026

Builds upon nAIVE Engine PRD v1.0

Table of Contents

1. Executive Summary

nAIVE Engine v2.0 represents a paradigm leap beyond the foundational architecture established in v1.0. Where v1.0 proved that AI could serve as a first-class authoring tool for game development — integrating Claude via MCP, Gaussian splatting as a native primitive, and YAML-driven declarative design — v2.0 asks a fundamentally bolder question: what happens when AI becomes a runtime component of the engine itself?

This document captures the evolutionary vision for nAIVE as a living, self-modifying engine where artificial intelligence is not merely a development tool but an active participant in every layer of the runtime stack — from procedural world generation to NPC cognition, from adaptive game direction to learnable rendering pipelines.

1.1 What Changed from v1.0 to v2.0

v1.0 established a solid foundation: Rust + wgpu + SLANG + hecs + Lua + YAML. v2.0 does not replace this stack — it extends it with eight evolutionary capabilities:

- **Self-Modifying Engine:** Dynamic Rust plugin architecture that allows the engine to hot-load, unload, and recompose its own subsystems at runtime.
- **LLM-Powered NPC Agents:** Non-player characters driven by language models with memory, personality, and genuine conversational ability — not finite state machines.
- **Natural Language Game Creation:** Players and designers speak games into existence through a natural language front-end that compiles intent into YAML scene graphs and Lua scripts.
- **Adaptive Game Evolution:** An AI Director that observes player telemetry in real-time and rewrites game rules, encounters, and narrative arcs mid-session.
- **Generative Gaussian Splatting:** Text-to-3DGS pipeline that generates photorealistic 3D assets from natural language descriptions — no photogrammetry required.
- **Learnable Render Pipeline:** Neural shader stages trained via gradient descent, enabling the render pipeline to learn aesthetic styles rather than having them hand-coded.
- **Multi-Agent Collaboration:** Specialized AI agent teams (level designer, narrative writer, balance tester, QA agent) that collaborate to build and refine games autonomously.
- **Web-Native Platform:** Browser-based game creation and play via WebGPU, making nAIVE a platform anyone can access without installation.

1.2 Vision Statement

nAIVE v2.0 is an engine where every traditional boundary — between author and player, between tool and runtime, between coded behavior and emergent intelligence — dissolves. Games built with nAIVE don't just run; they think, adapt, and evolve.

2. Architecture Overview

The v2.0 architecture retains the v1.0 core while introducing a new AI Runtime Layer that sits alongside the traditional game loop.

2.1 Core Stack (Retained from v1.0)

Layer	Technology	Purpose
Language	Rust (2021 edition)	Memory safety, zero-cost abstractions, fearless concurrency
Graphics	wgpu 24.x	Cross-platform GPU abstraction (Vulkan/Metal/DX12/WebGPU)
Shaders	SLANG → WGSL	High-level shader authoring with cross-compilation
ECS	hecs	Lightweight archetypal entity-component-system
Physics	Rapier 3D	Rigid body dynamics, collision detection
Audio	Kira	Game audio with real-time parameter control
Scripting	Lua 5.4 (mlua)	Gameplay scripting, hot-reload capable
Data Format	YAML	Scenes, materials, pipelines, events, input bindings
3D Primitives	Gaussian Splatting	First-class .ply splat rendering alongside traditional meshes

2.2 New: AI Runtime Layer (v2.0)

The AI Runtime Layer is a new architectural tier that operates in parallel with the traditional game loop. It consists of:

System	Technology	Purpose
Agent Runtime	LLM API + Local Models	NPC cognition, dialogue, decision-making
AI Director	Telemetry + LLM Reasoning	Real-time game adaptation based on player behavior
NL Compiler	Claude API + YAML Gen	Natural language to scene graph compilation
Gen-3DGS	Diffusion + 3DGS Pipeline	Text-to-Gaussian-splat asset generation
Neural Shaders	Differentiable Rendering	Learnable render pipeline stages
Plugin Host	Dynamic Rust Loading	Hot-loadable engine subsystem modification

Agent Swarm	Multi-Agent Orchestration	Specialized AI teams for collaborative development

2.3 System Architecture Diagram

The following layered architecture shows how v2.0 systems integrate with the v1.0 core:

```
┌──────────────────────────────────────────────────────────┐
│                  PLAYER / DESIGNER LAYER                 │
│   Natural Language ←→ WebGPU Browser ←→ CLI / MCP       │
├──────────────────────────────────────────────────────────┤
│               AI RUNTIME LAYER (v2.0 NEW)                │
│  Agent Runtime │ AI Director │ NL Compiler │ Gen-3DGS    │
│  Neural Shaders │ Plugin Host │ Agent Swarm              │
├──────────────────────────────────────────────────────────┤
│              ENGINE CORE LAYER (v1.0 RETAINED)           │
│  wgpu Renderer │ hecs ECS │ Rapier Physics │ Kira Audio │
│  SLANG Shaders │ Lua Scripts │ YAML Data │ Splat Render │
├──────────────────────────────────────────────────────────┤
│                     PLATFORM LAYER                       │
│    macOS (Metal) │ Linux (Vulkan) │ Web (WebGPU)         │
└──────────────────────────────────────────────────────────┘
```

3. Self-Modifying Engine Architecture

The most fundamental evolution in v2.0 is that the engine can modify itself at runtime. Rather than a monolithic binary, nAIVE becomes a dynamic composition of hot-loadable Rust plugins that can be swapped, upgraded, and reconfigured without restarting.

3.1 Dynamic Plugin System

Every major engine subsystem (renderer, physics, audio, AI) is compiled as a dynamic library (.dylib/.so/.dll) conforming to a stable ABI boundary. The plugin host manages lifecycle:

- **Discovery:** Plugins register via a manifest YAML file declaring their capabilities, dependencies, and interface version.
- **Hot-Loading:** The plugin host monitors a plugins/ directory and can load/unload plugins at runtime using Rust's libloading crate.
- **Versioning:** Semantic versioning ensures ABI compatibility. The host rejects plugins with incompatible interface versions.
- **Sandboxing:** Plugins operate within a capability-based security model — they can only access ECS components and engine APIs they explicitly declare.

3.2 Plugin Interface Specification

Each plugin implements a standard trait boundary:

```
pub trait NaivePlugin: Send + Sync {
    fn name(&self) -> &str;
    fn version(&self) -> SemVer;
    fn capabilities(&self) -> Vec<Capability>;
    fn init(&mut self, ctx: &mut EngineContext);
    fn update(&mut self, ctx: &mut EngineContext, dt: f64);
    fn shutdown(&mut self);
}
```

3.3 Self-Modification Scenarios

This architecture enables scenarios impossible with traditional engines:

- **AI-Driven Optimization:** The engine profiles its own performance and asks an LLM to suggest plugin configuration changes, then applies them live.
- **Genre Transformation:** Swapping the physics plugin from realistic Rapier to a stylized 2D plugin mid-game to shift genres.
- **Research Experimentation:** Researchers can swap renderer implementations (rasterization vs ray tracing vs neural rendering) without rebuilding.

4. LLM-Powered NPC Agent System

v2.0 replaces traditional finite state machines and behavior trees with a cognitive agent architecture where each NPC is backed by a language model with persistent memory, personality, and genuine understanding of its world.

4.1 Agent Architecture

Each NPC agent consists of four subsystems:

Subsystem	Implementation	Function
Perception	ECS query + spatial awareness	What the NPC can see, hear, and sense from the game world
Memory	Vector DB (local) + episodic log	Short-term working memory and long-term episodic recall
Reasoning	LLM inference (local or API)	Decision-making, dialogue generation, goal planning
Action	ECS component mutation	Translates LLM decisions into game-world actions

4.2 Personality System

NPC personalities are defined in YAML personality profiles that constrain and guide the LLM's responses:

```yaml
# character_profiles/merchant_elena.yaml
name: Elena the Merchant
traits: [shrewd, warm, superstitious]
knowledge: [trade_routes, local_history, herbalism]
goals: [profit, protect_family, find_lost_artifact]
speech_style: formal_but_friendly
memory_priority: [betrayals, kindness, rare_items]
```

4.3 Performance Budget

LLM inference is expensive. The agent system manages this through a tiered approach:

- **Tier 1 — Full LLM (conversation):** Used when the player directly engages an NPC. Latency budget: 500ms. Uses Claude API or a capable local model.
- **Tier 2 — Distilled Model (ambient behavior):** A smaller fine-tuned model handles routine NPC decisions (patrolling, trading, reacting to weather). Latency budget: 50ms.
- **Tier 3 — Cached Responses (background):** NPCs far from the player use pre-computed behavior patterns with personality-flavored variation. Latency budget: 1ms.

5. Natural Language Game Creation

The most user-facing revolution in v2.0: players and designers can describe what they want in plain English, and the engine compiles that intent into playable game content.

5.1 The NL Compiler Pipeline

Natural language input flows through a multi-stage compilation pipeline:

1. Intent Parsing: The user's natural language is analyzed by an LLM to extract structured intent (what kind of thing, where, with what properties).
2. Schema Validation: The extracted intent is validated against the nAIVE component catalog to ensure it maps to real engine capabilities.
3. YAML Generation: Valid intent is compiled into nAIVE YAML scene graph fragments — the same format used by hand-authored content.
4. Lua Scaffolding: If behavioral logic is implied (e.g., 'a door that opens when you bring the red key'), corresponding Lua scripts are generated.
5. Asset Resolution: If the description implies visual assets, the Gen-3DGS pipeline (Section 7) is triggered to create them.
6. Preview and Iteration: The generated content is placed into the scene for immediate preview, and the user can refine via further natural language.

5.2 Example Interactions

Designer: "Create a medieval tavern with a grumpy bartender, three tables, and a fireplace. The bartender should refuse to serve anyone who hasn't completed the forest quest."

nAIVE generates: Scene YAML with tavern geometry (via Gen-3DGS), entity definitions for bartender (with LLM agent personality: grumpy), table entities, fireplace with particle system and point light, and a Lua script that checks the player's quest_log component for forest_quest_complete before enabling the bartender's serve interaction.

Player (in-game): "I want to build a castle on that hill."

nAIVE generates: Castle splat asset via Gen-3DGS placed at the terrain point the player indicated, with appropriate collision volumes and structural integrity simulation.

5.3 Constraint System

Not everything should be possible. The NL compiler respects a constraint YAML that defines what the current game context allows:

- **World Rules:** Physics constraints, genre-appropriate assets, thematic consistency.
- **Balance Rules:** Prevents generating overpowered items, ensures economy coherence.
- **Narrative Rules:** Maintains story consistency, prevents plot contradictions.
- **Performance Rules:** Caps entity count, polygon budget, active agent count per scene.

6. Adaptive Game Evolution — The AI Director

The AI Director is a meta-system that observes everything happening in the game and intervenes to make the experience better. Unlike Left 4 Dead's director (which adjusted spawn rates), nAIVE's AI Director can rewrite game rules, restructure encounters, alter narrative arcs, and even modify the world's geography.

6.1 Telemetry Ingestion

The Director consumes a real-time telemetry stream from the ECS:

- **Player State:** Position, health, inventory, quest progress, combat stats, time-in-area.
- **Emotional Proxies:** Movement speed (rushing vs exploring), combat aggression, dialogue choices, retry count, session duration.
- **World State:** NPC populations, resource availability, environmental conditions, active quests.
- **Meta-Patterns:** Play style classification (explorer, fighter, builder, socializer), skill trajectory, engagement curves.

6.2 Intervention Capabilities

Based on its analysis, the Director can take graduated actions:

Level	Intervention	Example
Subtle	Parameter tuning	Increase loot drop rates for struggling player
Moderate	Content injection	Spawn a helpful NPC with a hint when player is stuck
Significant	Encounter redesign	Replace boss fight with puzzle for non-combat player
Major	Narrative branching	Generate a new quest line based on player's emergent behavior
Transformative	World restructuring	Reshape terrain, add new regions, evolve the world based on collective play

6.3 Ethical Guardrails

The AI Director operates within strict ethical boundaries:

- Transparency: Players can toggle a 'Director Notes' overlay to see why changes were made.
- Consent: Major interventions (narrative changes, world restructuring) require player opt-in.
- No Addiction Mechanics: The Director optimizes for engagement and satisfaction, never for compulsive behavior loops.

- Fairness: In multiplayer contexts, the Director ensures interventions don't create unfair advantages.

7. Generative Gaussian Splatting Pipeline

v1.0 established Gaussian splatting as a first-class rendering primitive, loading pre-captured .ply splat data. v2.0 makes splatting generative — assets are synthesized from text descriptions in real-time, bypassing the traditional photogrammetry capture pipeline entirely.

7.1 Pipeline Architecture

7. Text Encoding: Natural language description is embedded using a CLIP-style encoder that understands both semantic meaning and spatial relationships.
8. Multi-View Diffusion: A diffusion model generates consistent multi-view images from the text embedding (similar to Zero123++ or MVDream architecture).
9. 3DGS Reconstruction: The multi-view images are fed into a rapid 3DGS reconstruction module that produces a .ply splat cloud.
10. Optimization: A fast differentiable splat optimizer refines Gaussian positions, scales, rotations, and spherical harmonics for the target quality level.
11. LOD Generation: Multiple levels of detail are generated by progressive Gaussian pruning.
12. Integration: The final splat is registered into the scene graph with appropriate collision bounds auto-generated from the splat density field.

7.2 Quality Tiers

Tier	Gaussians	Gen Time	Use Case	Quality
Preview	~10K	2-5 seconds	Rapid iteration	Draft
Standard	~100K	15-30 seconds	Gameplay assets	Good
High	~500K	1-3 minutes	Hero assets	Excellent
Cinematic	~2M	5-15 minutes	Cutscenes, key props	Photorealistic

7.3 Style Transfer

Generated splats can be style-transferred to match a target aesthetic. A designer can provide a reference image or style description (e.g., 'Studio Ghibli watercolor', 'pixel art voxels', 'dark souls gothic'), and the optimization phase will guide the splat's spherical harmonic coefficients toward that visual style while preserving geometric fidelity.

8. Learnable Render Pipeline

The most technically ambitious evolution: portions of the render pipeline are replaced with differentiable, trainable neural network stages that can learn visual styles instead of having them hand-coded in shaders.

8.1 Differentiable Rendering Architecture

The v2.0 render pipeline DAG (retained from v1.0's YAML-defined pass graph) is extended to support neural pass nodes alongside traditional SLANG shader passes:

- **Traditional Passes:** Geometry, lighting, shadow maps, Gaussian splat rendering — these remain SLANG-based for predictable, real-time performance.
- **Neural Passes:** Post-processing, style transfer, ambient occlusion estimation, denoising, super-resolution — these are neural network stages that run as compute shaders with learned weights.
- **Hybrid Passes:** Some stages blend both approaches, using neural networks to guide traditional rendering (e.g., neural importance sampling for ray tracing).

8.2 Training Pipeline

Neural shader stages are trained offline and deployed as frozen weight files:

13. Reference Rendering: A high-quality reference is produced (path-traced, high sample count, or artist reference images).
14. Neural Stage Training: The neural pass is trained via gradient descent to approximate the reference output from the inputs available at its pipeline position.
15. Quantization: Weights are quantized to FP16 or INT8 for real-time inference on GPU compute shaders.
16. Deployment: The trained weights are packaged as a .neural_shader asset and referenced in the render pipeline YAML.

8.3 YAML Pipeline Extension

```
# render_pipeline.yaml (v2.0 extension)
passes:
  - name: geometry_pass
    type: raster          # traditional
    shader: geometry.slang
  - name: style_transfer
    type: neural           # NEW in v2.0
    weights: ghibli_style.neural_shader
    inputs: [color_buffer, depth_buffer]
    output: styled_color
```

9. Multi-Agent Collaborative Development

v1.0 introduced MCP integration for a single Claude instance to interact with the engine. v2.0 scales this to a swarm of specialized AI agents that collaborate to build, test, and refine games.

9.1 Agent Roles

Agent	Responsibility	Tools / Access
Level Architect	Designs spatial layouts, generates terrain, places landmarks	NL Compiler, Gen-3DGS, Scene YAML
Narrative Director	Writes quests, dialogue trees, NPC personalities, story arcs	Character YAML, Lua scripts, NPC agents
Systems Designer	Creates game mechanics, economy rules, progression systems	Component definitions, Lua, balance data
QA Tester	Plays the game autonomously, reports bugs, edge cases, balance issues	Full game API, telemetry, bug tracker
Art Director	Ensures visual consistency, guides style transfer, reviews assets	Neural shaders, Gen-3DGS, materials YAML
Orchestrator	Coordinates other agents, resolves conflicts, manages priorities	All agent APIs, project state, user preferences

9.2 Collaboration Protocol

Agents communicate through a shared message bus with structured proposals:

17. An agent proposes a change (e.g., Narrative Director: 'Add a blacksmith NPC to the village with a redemption quest arc').
18. Affected agents review the proposal (Level Architect checks spatial fit, Systems Designer checks economy impact, Art Director checks visual consistency).
19. Conflicts are resolved by the Orchestrator, which can escalate to the human designer for final decisions.
20. Approved changes are atomically committed to the scene graph.
21. The QA Tester plays through affected areas and reports any regressions.

9.3 Human-in-the-Loop

The human designer remains the ultimate authority. The agent swarm presents its work as proposals that can be accepted, modified, or rejected. The system learns from these decisions to better anticipate the designer's preferences over time.

10. Web-Native Platform

The boldest distribution decision in v2.0: nAIVE ships as a web experience. No download, no installation, no platform lock-in. Open a browser, start creating.

10.1 WebGPU Target Architecture

wgpu already compiles to WebGPU via wasm-bindgen. v2.0 makes this the primary distribution target:

- **Rust to WASM:** The entire engine core compiles to WebAssembly via wasm-pack, producing a .wasm binary that runs in any WebGPU-capable browser.
- **SLANG to WGSL:** SLANG shaders cross-compile to WGSL (already supported in v1.0).
- **Asset Streaming:** Large assets (splats, textures, neural shader weights) are streamed progressively rather than loaded upfront.
- **Compute Budget:** AI inference runs on the GPU via WebGPU compute shaders or falls back to server-side API calls for complex reasoning.

10.2 Platform Capabilities

Feature	Browser Implementation	Native Equivalent
Rendering	WebGPU	wgpu (Vulkan/Metal/DX12)
Compute	WebGPU Compute Shaders	Native Compute
Storage	Origin Private File System	Local filesystem
Networking	WebSockets + WebRTC	TCP/UDP
AI Inference	WebGPU + Server Fallback	Local GPU + API
Audio	Web Audio API	Kira

10.3 Sharing and Collaboration

Web-native means every game created with nAIVE is instantly shareable via URL. A designer creates a game, gets a link, and anyone can play it in their browser. This transforms nAIVE from a tool into a platform — the YouTube of game creation.

11. Retained Specifications from v1.0

The following v1.0 specifications remain fully in effect and are not reproduced here in full. Refer to nAIVE Engine PRD v1.0 for complete details:

11.1 YAML Format Specifications

- **Scene Format:** Entity definitions with components, inheritance via _template, nested children.
- **Material Format:** Shader references, texture bindings, uniform parameters, blend modes.
- **Render Pipeline Format:** DAG of passes with inputs/outputs/dependencies (extended in v2.0 with neural pass type).
- **Event System:** Event definitions with typed payloads, handler bindings.
- **Input Binding:** Context-based input mapping with action/axis types.

11.2 Component Catalog

All v1.0 components remain valid. v2.0 adds new components:

New Component	Purpose
agent_brain	Attaches an LLM agent to an entity with personality profile, memory config, and inference tier.
director_target	Marks an entity as subject to AI Director intervention with intervention level cap.
generated_asset	Tracks provenance of procedurally generated assets (text prompt, generation params, quality tier).
nl_editable	Marks an entity as modifiable via natural language commands in play mode.
network_replicated	Marks an entity for network replication in multiplayer web sessions.

11.3 MCP Server (Extended)

The v1.0 MCP server is extended with new tool endpoints for v2.0 capabilities:

- **naive/agent/create:** Spawn a new LLM NPC agent with personality profile.
- **naive/agent/converse:** Initiate conversation with an NPC agent.
- **naive/director/status:** Query the AI Director's current analysis and planned interventions.
- **naive/generate/splat:** Request Gen-3DGS asset creation from text description.
- **naive/nl/compile:** Submit natural language for scene graph compilation.
- **naive/plugin/list:** List loaded plugins and their status.
- **naive/plugin/swap:** Hot-swap a plugin at runtime.

12. Implementation Roadmap

v2.0 is designed as a layered extension of v1.0. The user is already building v1.0 (20-week plan). v2.0 features are organized into three horizons that can begin once v1.0 core is stable.

12.1 Horizon 1: Foundation Extensions (Weeks 21-32)

These features build directly on v1.0 infrastructure with minimal architectural risk:

Phase	Feature	Duration	Dependencies
H1.1	Dynamic Plugin System — ABI boundary, hot-loading, manifest format	3 weeks	v1.0 Phase 1-2 complete
H1.2	NPC Agent Prototype — Single LLM agent with personality, memory, perception	4 weeks	v1.0 ECS + Lua
H1.3	NL Compiler v1 — Text to YAML scene generation via Claude API	3 weeks	v1.0 MCP server
H1.4	WebGPU Build Target — WASM compilation, browser asset loading	2 weeks	v1.0 wgpu renderer

12.2 Horizon 2: Intelligence Layer (Weeks 33-48)

These features introduce the AI runtime layer as a first-class engine tier:

Phase	Feature	Duration	Dependencies
H2.1	AI Director — Telemetry ingestion, intervention system, ethical guardrails	4 weeks	H1.2 agent prototype
H2.2	Gen-3DGS Pipeline — Text-to-splat generation with quality tiers	4 weeks	v1.0 splat renderer
H2.3	Multi-Agent Orchestration — Agent roles, collaboration protocol, human-in-loop	4 weeks	H1.2, H1.3
H2.4	Neural Shader Prototype — Single differentiable post-process pass, training pipeline	4 weeks	v1.0 render pipeline DAG

12.3 Horizon 3: Platform (Weeks 49-64)

These features transform nAIVE from a tool into a platform:

Phase	Feature	Duration	Dependencies
H3.1	Web Platform — URL sharing, collaborative editing, asset CDN	4 weeks	H1.4 WebGPU target

H3.2	Player Creation Mode — In-game NL creation, constraint system, UGC tools	4 weeks	H1.3, H2.2
H3.3	Self-Modifying Engine — AI-driven plugin optimization, genre transformation	4 weeks	H1.1, H2.1
H3.4	Demo Game: Relic v2.0 — Full showcase integrating all v2.0 capabilities	4 weeks	All above

13. Demo Game: Relic v2.0

v1.0's Relic demo (dungeon crawler) is reimagined to showcase every v2.0 capability:

13.1 What's New in Relic v2.0

- **Generative Dungeon:** Each play session generates a unique dungeon via Gen-3DGS — no two runs look the same. Describe your ideal dungeon aesthetic before entering ('obsidian cavern with bioluminescent fungi' or 'ancient Roman catacombs').
- **Conversational NPCs:** The merchant, quest-giver, and rival adventurer are all LLM agents with persistent memory. They remember your previous runs, hold grudges, form alliances, and gossip about you to other NPCs.
- **Adaptive Challenge:** The AI Director monitors your combat performance, exploration patterns, and puzzle-solving speed. It redesigns encounters mid-session — if you're breezing through combat, it introduces environmental puzzles; if you're stuck, it sends a helpful NPC with a cryptic hint.
- **Player Creation:** Between dungeon runs, players can modify their home base using natural language ('add a trophy wall for my defeated bosses' or 'build a garden with that crystal I found'). These modifications persist across sessions.
- **Shared Web Experience:** Relic runs entirely in the browser. Players can share their dungeon seeds and home bases via URL. A spectator mode lets friends watch live runs.
- **Style Modes:** Neural shader stages let players switch visual style mid-run — photorealistic, pixel art, ink wash painting, or comic book cel shading — without any asset changes.

14. Risk Assessment and Mitigation

Risk	Severity	Mitigation	Fallback
LLM latency too high for real-time NPC	High	Tiered inference with local distilled models	Sophisticated behavior trees with LLM-generated dialogue cache
Gen-3DGS quality insufficient	Medium	Progressive quality tiers, offline generation option	Curated splat library with style transfer
WebGPU adoption incomplete	Medium	WebGL2 fallback renderer (reduced features)	Native-first distribution with web as secondary
Neural shader perf on consumer GPU	High	Aggressive quantization, resolution scaling	Traditional SLANG shaders as default, neural as opt-in
Dynamic plugin ABI stability	Medium	Strict versioned trait interfaces, compatibility testing	Static linking with configuration-based feature toggling
AI Director player trust	Low	Transparency overlay, opt-in for major changes	Manual difficulty settings as traditional alternative
Multi-agent coordination coherence	Medium	Atomic proposals, human approval for conflicts	Single-agent mode with sequential task processing
LLM API cost at scale	High	Local model inference, aggressive caching, batch processing	Hybrid approach: local models for routine, API for complex reasoning

15. Success Metrics

15.1 Technical Metrics

Metric	Target	Measurement
Frame rate with neural shaders	>30 FPS on M1 MacBook	Automated benchmark suite
NPC response latency (Tier 1)	<500ms for conversation	P95 response time logging
Gen-3DGS preview generation	<5 seconds for 10K Gaussians	End-to-end pipeline timing
NL Compiler accuracy	>80% first-attempt correct scene gen	Human evaluation of generated scenes
WASM binary size	<50MB initial load	Build output measurement
Plugin hot-swap time	<100ms with no frame drops	Plugin lifecycle benchmark

15.2 Experience Metrics

- A non-programmer can create a playable game scene in under 5 minutes using only natural language.
- NPC conversations feel meaningfully different across multiple play sessions (memory persistence validated).
- AI Director interventions are rated 'helpful' or 'invisible' by >70% of playtesters (never 'annoying' or 'unfair').
- Relic v2.0 demo is playable entirely in a browser with no installation step.
- Style-transferred rendering is visually indistinguishable from purpose-built art assets in blind tests.

16. Conclusion

nAIVE Engine v2.0 is not merely an upgrade — it is a thesis about what game engines should become when artificial intelligence matures from a development tool into a runtime primitive. Every design decision in this document points toward a single conviction: the most interesting games of the next decade will not be authored — they will be grown, evolved, and co-created by humans and AI working together in real-time.

The v1.0 foundation (Rust + wgpu + SLANG + hecs + Lua + YAML + Gaussian splatting + MCP) is not replaced; it is proven infrastructure upon which the eight evolutionary capabilities of v2.0 are built. The implementation roadmap is deliberately layered: each horizon delivers independently valuable functionality while building toward the full platform vision.

The question nAIVE v2.0 answers is not 'can AI help make games?' — that question was answered in v1.0. The question is: 'what becomes possible when the AI is a living part of the game itself?' The answer, we believe, is an entirely new category of interactive experience that could not exist before.

— End of Document —