Generative Worldcrafting: A Modular Framework for AI-Assisted Content Creation in Games

Tomislav Peharda, Bogdan Okreša Đurić
University of Zagreb Faculty of Organization and Informatics
Artificial Intelligence Laboratory
Pavlinska 2, 42000 Varaždin, Croatia
{tpeharda, dokresa}@foi.unizg.com

Abstract. *We present a modular framework for AI-assisted world generation in sandbox games, using Minecraft as a case study. Our system integrates multimodal user inputs, i.e. text, speech, and images, with large language and vision-language models to generate structured in-game content. By decomposing complex prompts into reusable components and translating them into executable game commands, the approach supports scalable and intuitive content creation. This work demonstrates how generative AI can enhance creative expression and interactivity in virtual environments.*

Keywords. generative artificial intelligence, artificial intelligence, video games, virtual world generation, large language models, Minecraft, multimodal interface

1 Introduction

Recent advancements in generative artificial intelligence (Gen AI) have opened up new possibilities across a wide range of domains, with the game industry being one of the areas of application (Jovanovic & Campbell, 2022; Takale et al., 2024). One compelling use case is the generation of virtual worlds, where the stochastic nature of generative models aligns well with the need for creativity and variability. Unlike domains that require deterministic outputs, video games thrive in environments where multiple outcomes are not only acceptable but encouraged. This makes the game industry an ideal candidate for the deployment of Gen AI technologies (Marr, 2024).

Players have consistently demonstrated a strong interest in shaping their own virtual experiences, as evidenced by the popularity of sandbox and builder games such as Minecraft (Mojang Studios, 2011), Roblox (Roblox Corporation, 2025), and similar platforms (Carbonell-Carrera et al., 2021). Despite their creative freedom, these games often rely on user interfaces that demand extensive manual interaction, typically involving keyboard and mouse inputs. This requirement can pose a barrier to seamless and intuitive world-building experiences.

This research explores the intersection of Gen AI and virtual world generation by reviewing existing academic and industry work in the field. It also proposes practical approaches for leveraging large language models (LLMs) to generate rich, dynamic game content. Furthermore, the study outlines how these models can be integrated to power world generation in existing game engines and workflows, potentially transforming how users interact with and create within virtual environments.

2 Related work

Gen AI refers to computational techniques that produce new and meaningful content, such as text, images, or audio, by learning from large datasets (Feuerriegel et al., 2024). Unlike traditional models focused on classification, generative models aim to understand and replicate data distributions, enabling them to generate realistic outputs. A prominent example are LLMs, such as GPT-3 and GPT-4, which are built on transformer architectures and trained on extensive text corpora (Fui-Hoon Nah et al., 2023). These models are capable of handling a wide variety of language tasks, including dialogue, summarization, and creative writing, often with little or no task-specific training. Recently, visual large language models (vLLMs) have extended this capability to multi-modal domains by processing both visual and textual information. vLLMs can interpret images and generate coherent text based on visual input, enabling tasks like image captioning, visual question answering, and multimodal reasoning. Their flexibility and ability to generalize across domains have made them foundational tools in many modern artificial intelligence (AI) applications.

Speech-to-text (STT) models, which can also be considered part of Gen AI, are systems that convert spoken language into written text by analyzing audio signals and mapping them to linguistic representations (Reddy et al., 2023; Trivedi et al., 2018). These models are trained on large datasets of speech and corresponding transcripts, allowing them to learn the patterns of

spoken language across different accents, tones, and contexts. While their primary function is transcription rather than creative generation, they play a crucial role in multimodal generative systems by enabling audio-based interaction. For example, they are often used as the first step in voice-controlled applications or conversational agents, where spoken input is transcribed and then passed to a language model for further processing.

BRICKGPT is a generative model that creates physically stable LEGO-style structures from text prompts (Pun et al., 2025). It fine-tunes a large language model to predict brick placements step by step, using a custom dataset called StableText2Brick. To ensure the designs are buildable, it applies physics-based validity checks and removes unstable bricks during generation. The outputs can be assembled manually or by robots and support color and texture customization from descriptive text.

SpAItial AI (SpAItial AI, 2025) is developing a new class of generative models called Spatial Foundation Models, which are designed to reason about and generate three-dimensional environments grounded in physical space and time. Unlike conventional generative models that operate in image space or use 2D projections, these models work directly in 3D coordinate systems, enabling consistent and coherent representations of objects, scenes, and environments. The goal is to enable machines to understand and interact with the physical world in a spatially aware manner, supporting applications such as digital twinning, autonomous systems, and 3D content generation with high fidelity and structural accuracy.

VOYAGER is an open-ended, LLM-powered lifelong learning agent (Park et al., 2023) designed to operate in the game Minecraft. It uses GPT-4 to explore, acquire skills, and solve increasingly complex tasks without human intervention. VOYAGER features three core components: an automatic curriculum that proposes tasks based on exploration progress, a skill library for storing and retrieving reusable action programs, and an iterative prompting mechanism that incorporates environment feedback and execution errors to refine its code. This design allows VOYAGER to learn and adapt continuously, outperforming other methods in exploration, task completion, and generalization to new scenarios.

Genie 2 is a world model by DeepMind that generates interactive 3D environments from a single image prompt (Lillicrap et al., 2024). It allows users or AI agents to explore these worlds using keyboard and mouse, with consistent physics, object behavior, and memory. Built on a video diffusion model, Genie 2 predicts each frame based on actions, enabling real-time, controllable gameplay suitable for training agents and creative applications.

Among many video games that offer the ability to build worlds, one of the most popular ones is Minecraft (Mojang Studios, 2011). Minecraft is a sandbox builder game where players can create and explore environments made entirely of blocks. It provides a wide range of materials and tools that players use to shape the world, from constructing simple homes to designing entire cities or landscapes. With its open-ended gameplay, Minecraft encourages creativity by letting users build their own worlds from the ground up using the blocks they're given (Carbonell-Carrera et al., 2021).

3 Proposed Approach

Building on prior related work, this research focuses on proposing approaches for utilizing Gen AI to build worlds within existing video games, using Minecraft as an example. The approach to generation is bottom-up, where the user starts by generating individual objects that make up the world.

The three key components of this proposal are:

- The user communication interface,
- The world generation business logic,
- The interface to the game engine.

The communication interface refers to the visual, textual, or audio modalities through which users describe the objects they wish to generate. Inputs received through this interface are transformed into corresponding actions within the world generation business logic, powered by an LLM, and are subsequently used by the interface to the game engine to trigger appropriate actions for generating the world inside the game.

3.1 Communication Interfaces

This paper investigates multiple user interfaces for interacting with the system and translating user inputs into game actions. These include textual, audio, and visual modalities, each enabling intuitive and flexible interaction patterns for different user preferences or accessibility needs.

The textual interface allows users to input natural language commands, such as "generate a house with two floors and a garden", which are directly processed by an LLM. This is the most straightforward modality and serves as the foundation for others. The audio interface leverages STT models to transcribe spoken commands into text. These models are integrated upstream of the LLM pipeline, allowing seamless conversion of voice commands into structured text instructions. This approach makes the system usable in hands-free settings.

The visual interface utilizes vLLM, capable of interpreting user-submitted images or drawings. These models convert visual stimuli into textual descriptions that serve as LLM prompts. For instance, if a user uploads a sketch of a medieval tower, vLLM interprets the drawing and produces a textual description such

as “generate a tall stone tower with wooden windows and a red flag”. This modality enables novel and intuitive design workflows, where players can create game structures through sketches or visual references instead of verbal instructions.

Listing 1 features an example prompt that could be passed to a vLLM alongside the image, in order to generate an instruction.

Listing 1. vLLM sample prompt to analyze image and generate textual instruction

```
# Role
You are an expert at analyzing images and generating a textual instruction on what is in the image.

# Instructions
1. Analyze what are the objects you see on the image
2. Generate a description of what you see as an instruction of what needs to be generated

# Example
"Generate a house with 5 windows and 1 door"
```

By supporting multimodal interaction, the system aims to make generative world-building accessible, expressive, and personalized.

3.2 Business Logic for World Generation

Once the user input is transformed into a textual query, the system’s core logic is responsible for interpreting the prompt and generating corresponding game actions. This logic is built around the capabilities of LLMs to follow instructions, reason about structures, and generate sequences of domain-specific operations.

The LLM is prompted with detailed instructions and few-shot examples that teach it how to translate user intentions into actions tailored to the game interface. For example, the input "generate a house" would result in an ordered list of Minecraft block placements, structured by dimensions, materials, and spatial relationships.

Listing 2 shows an example system prompt to an LLM.

Listing 2. System prompt to produce world generation steps

```
# Role
You are an expert at generating game actions that produce blocks that the expected object is comprised of.

# Instructions
1. Analyze user message and determine what object needs to be created.
2. Each object is comprised of one or multiple small size box-like blocks
3. Generate blocks with coordinates that are needed to shape a 2D representation of the requested object
4. Output should be list of objects comprised of x and y coordinates

# Example
Input: "Generate a tree"
Output: [{"x": 0, "y": 0}, {"x": 0, "y": 1}, ...]
```

Listing 3 shows a sample expected output.

Listing 3. Object generation output

```
[
    {"x": 0, "y": 0}, {"x": 0, "y": 1}, {"x": 0, "y": 2}, {"x": 0, "y": 3}, # trunk
    {"x": -1, "y": 4}, {"x": 0, "y": 4}, {"x": 1, "y": 4}, # bottom foliage layer
    {"x": -1, "y": 5}, {"x": 0, "y": 5}, {"x": 1, "y": 5}, # middle foliage layer
    {"x": 0, "y": 6} # top foliage
]
```

To improve compositionality and reuse, an alternative approach is to have the LLM break down complex objects into known components. For instance, instead of generating an entire house from scratch, the model can output a set of required components such as windows and doors, each of which can be retrieved from a database.

An example system prompt to the LLM is shown in Lst. 4.

Listing 4. System prompt to break down complex object into components

```
# Role
You are an expert at providing list of components that are needed to generate a complex object, given the available components.

# Instructions
1. Analyze user message and determine what are the components that the expected object given the list of available components
2. Provide an output as a list of unique strings that describe needed components

# Available components
- window
- door
- roof

# Example
Input: "Generate a house with 5 windows and 2 doors"
Output: ["window", "door"]
```

That being said, if the user query is “Generate a house with 5 windows and 2 doors,” the LLM output would be: `window` and `door`. These two components are then searched against a database where the required blocks for generating each object are stored. Upon retrieval, the LLM can be queried again, this time with the sets of blocks for both components, asking it to generate the originally requested object using those components. An example LLM prompt is shown in Lst. 5.

Listing 5. System prompt to synthesis low-level object blocks into a complex object

```
# Role
You are an expert at producing steps to build a complex object given the available sub-components and its corresponding blocks.

# Instructions
1. Analyze what are the components that a complex object is comprised of.
2. Components to build a complex object must be from the list of available components
3. Utilize the blocks for building low-level components to build complex object

# Available components
```

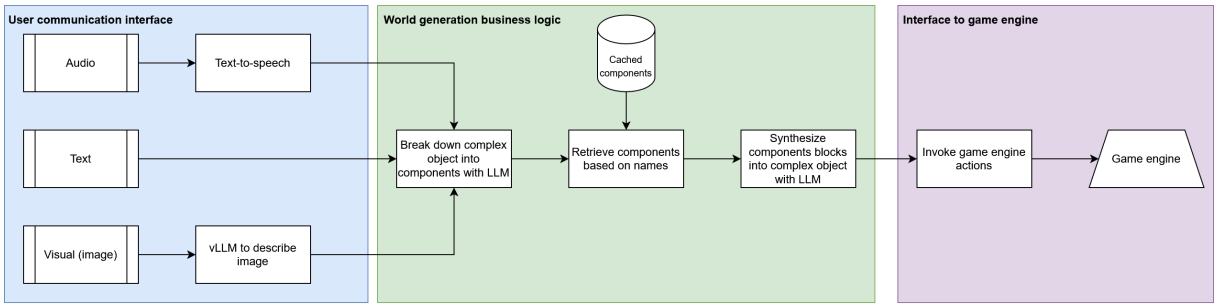

Figure 1. Complex object generation process

```
- window: [<list of objects with x and y ↩
  coordinates to build window>]
- door: [<list of objects with x and y ↩
  coordinates to build door>]

# Example
Input: "Generate a house with 5 windows and 2 ↩
  doors"
Output: [{"x": 0, "y": 0}, {"x": 0, "y": 1}, ↩
  ...]
```

This component-based generation can be formalized as a three-step process:

1. Use the LLM to decompose a complex object into its constituent parts or components.
2. Perform a search over a database of predefined generation blocks for these components.
3. Return the retrieved instructions to the LLM and ask it to synthesize them into a complex object.

This hybrid reasoning framework supports both creative and reusable generation. It benefits from the LLM's ability to generalize, while maintaining consistency and efficiency through the reuse of known building blocks.

3.3 Interface to a Game Engine

The final component of the system ensures that the generated world representations are rendered within the game environment. This requires an execution interface that connects the LLM's outputs to the concrete API of the game engine.

This interface can take the form of a custom-built socket server, a game engine plugin, or an intermediary software development kit. Its purpose is to consume the LLM-generated instructions and translate them into executable commands in the target game. For instance, in Minecraft, this might involve calling Java APIs or sending commands via a server console to modify the game world in real-time.

This layer must also handle synchronization, error handling, and user feedback. It ensures that generated instructions conform to the game's capabilities and gracefully degrade when the output exceeds system constraints. Additionally, the interface could support real-time feedback loops, where the game engine returns information to the LLM or user. For instance, describing whether a structure was successfully built or if an object is invalid in the current context.

By closing the loop from user query to in-game realization, this interface enables dynamic, AI-driven content creation that feels native to the game environment.

Fig. 1 presents the process of breaking a complex object into components and utilizing them to generate blocks for its construction.

4 Conclusion

This work presents a modular framework for AI-assisted world generation, integrating multimodal user interfaces with the reasoning capabilities of large language models to generate structured game content. By decomposing complex user requests into reusable components and translating them into actionable game commands, the system supports both creative flexibility and structural consistency. The proposed approach demonstrates how generative models can enhance user interaction in sandbox environments like Minecraft, making content creation more intuitive and expressive.

Future work could explore the extension of this framework beyond 2D representations into full 3D spatial reasoning. This would involve adapting prompts and component libraries to support volumetric structures and spatial constraints more effectively. Additionally, integrating reinforcement learning or planning-based agents alongside LLMs may enable the system to validate and optimize generated structures in context. Finally, user evaluation studies could be conducted to assess the usability, creativity support, and overall engagement enabled by multimodal world generation systems.

References

Carbonell-Carrera, C., Jaeger, A. J., Saorín, J. L., Melián, D., & De la Torre-Cantero, J. (2021). Minecraft as a block building approach for developing spatial skills. *Entertainment Computing*, *38*, 100427.

Feuerriegel, S., Hartmann, J., Janiesch, C., & Zschech, P. (2024). Generative ai. *Business & Information Systems Engineering*, *66*(1), 111–126.

Fui-Hoon Nah, F., Zheng, R., Cai, J., Siau, K., & Chen, L. (2023). Generative ai and chatgpt: Applications, challenges, and ai-human collaboration.

Jovanovic, M., & Campbell, M. (2022). Generative artificial intelligence: Trends and prospects. *Computer*, *55*(10), 107–112.

Lillicrap, T., et al. (2024). Genie 2: A large-scale foundation world model [DeepMind blog].

Marr, B. (2024). The role of generative ai in video game development [Accessed: 2025-07-03]. *Forbes*. https://www.forbes.com/sites/bernardmarr/2024/04/18/the-role-of-generative-ai-in-video-game-development/

Mojang Studios. (2011). Minecraft.

Park, J. S., O'Brien, J., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. *Proceedings of the 36th annual acm symposium on user interface software and technology*, 1–22.

Pun, A., Deng, K., Liu, R., Ramanan, D., Liu, C., & Zhu, J.-Y. (2025). Generating physically stable and buildable lego designs from text. *arXiv preprint arXiv:2505.05469*.

Reddy, V. M., Vaishnavi, T., & Kumar, K. P. (2023). Speech-to-text and text-to-speech recognition using deep learning. *2023 2nd international conference on edge computing and applications (ICECAA)*, 657–666.

Roblox Corporation. (2025). Roblox [Accessed: 2025-07-03].

SpAItial AI. (2025, July). Announcing spaitial [Accessed: 2025-07-02].

Takale, D. G., Mahalle, P. N., & Sule, B. (2024). Advancements and applications of generative artificial intelligence. *Journal of Information Technology and Sciences*, *10*(1), 20–27.

Trivedi, A., Pant, N., Shah, P., Sonik, S., & Agrawal, S. (2018). Speech to text and text to speech recognition systems-areview. *IOSR J. Comput. Eng*, *20*(2), 36–43.