RESEARCH-ARTICLE

Conversational Interactions with Procedural Generators using Large Language Models

JIM WHITEHEAD, University of California, Santa Cruz, Santa Cruz, CA, United States

THOMAS WESSEL, University of California, Santa Cruz, Santa Cruz, CA, United States

BLYTHE CHEN, University of California, Santa Cruz, Santa Cruz, CA, United States

RAVEN CRUZ-JAMES, University of California, Santa Cruz, Santa Cruz, CA, United States

LUC HARNIST, University of California, Santa Cruz, Santa Cruz, CA, United States

WILLIAM KLUNDER, University of California, Santa Cruz, Santa Cruz, CA, United States

View all

Published: 15 April 2025

FDG '25: International Conference on the Foundations of Digital Games
April 15 - 18, 2025
Vienna & Graz, Austria

https://doi.org/10.1145/3723498.3723788
ISBN: 9798400718564

Conversational Interactions with Procedural Generators using Large Language Models

Jim Whitehead
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
ejw@ucsc.edu

Thomas Wessel
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
twessel@ucsc.edu

Blythe Chen
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
blschen@ucsc.edu

Raven Cruz-James
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
rcruzjam@ucsc.edu

Luc Harnist
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
lharnist@ucsc.edu

William Klunder
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
wklunder@ucsc.edu

Justin Lam
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
jlam53@ucsc.edu

Ethan Lin
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
elin38@ucsc.edu

Roman Luo
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
yluo111@ucsc.edu

Hung Nguyen
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
huminguy@ucsc.edu

Naitik Poddar
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
npoddar@ucsc.edu

Shiva Ravinutala
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
shravinu@ucsc.edu

Alejandro Montoreano
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
amontore@ucsc.edu

Logan Shehane
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
lshehane@ucsc.edu

Yazmyn Sims
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
yzsims@ucsc.edu

Jarod Spangler
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
jrspangl@ucsc.edu

Michelle Tan
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
mtan46@ucsc.edu

Zosia Trela
Dept. of Computational Media
University of California, Santa Cruz
Santa Cruz, CA, USA
ztrela@ucsc.edu

Abstract

This paper explores the potential of Large Language Models (LLMs) to facilitate conversational natural language interactions to aid humans in mixed-initiative generation of game worlds. This paper explores the issues in creating a system that allows for rapid iteration in a turn-based user-LLM design software. We identify key research topics, including game world representation in LLMs, natural language-based world manipulation using function calls, and direct manipulation from processed LLM output. We successfully created a QA dataset to compare the accuracy of leading models using text, images, or both. Our findings highlight the potential of LLMs to assist in procedural content generation through enhanced natural language interaction and conversations while revealing the challenges of in-game world manipulation that warrant further research.

FDG '25, Graz, Austria

ACM ISBN 979-8-4007-1856-4/25/04
https://doi.org/10.1145/3723498.3723788

CCS Concepts

• **Computing methodologies** → **Natural language processing**;
• **Applied computing** → **Computer games**.

This is a Corrected VoR published on December 26, 2025. The VoR may still be accessed via the Supplemental Material section at https://dl.acm.org/doi/10.1145/3723498.3723788

Keywords

Procedural content generation, large language models, natural language interaction, conversational interaction with procedural generators

ACM Reference Format:

Jim Whitehead, Thomas Wessel, Blythe Chen, Raven Cruz-James, Luc Harnist, William Klunder, Justin Lam, Ethan Lin, Roman Luo, Hung Nguyen, Naitik Poddar, Shiva Ravinutala, Alejandro Montoreano, Logan Shehane, Yazmyn Sims, Jarod Spangler, Michelle Tan, and Zosia Trela. 2025. Conversational Interactions with Procedural Generators using Large Language Models. In *International Conference on the Foundations of Digital Games (FDG '25), April 15–18, 2025, Graz, Austria.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3723498.3723788

1 Introduction

Procedural generators for games support the expressive goals of designers. From the serenely alien landscapes of No Man's Sky to the unpredictably deadly levels of Spelunky, procedural generators act to support a game's aesthetics, gameplay, and emotional tone. When communicating with other human team members, designers use a wide range of emotive terms to describe these expressive goals. Traditional procedural generation systems give designers a series of input parameters and expect them to understand the mapping between expressive goals and parameter spaces. In contrast, human languages permit expressive goals to be stated directly. Using natural language, a designer can request a generator to create an "angry ocean" or a "tranquil sea", instead of futzing with the parameters of a wave simulation model.

Historically, the technical complexity of natural language parsing and understanding has limited its use in procedural content generation systems. For example, Mobramaein created a functional natural language interface for the creation of ocean wave simulations in Unity games, but was limited to understanding only predefined language patterns and a fixed set of adjectives [12]. The lack of expressive natural language interfaces that can integrate into existing applications has limited the ability to create procedural generators with a large range of natural language inputs.

Large language models (LLMs) change this. LLMs can accept and understand an exceptionally wide range of natural language inputs. This ability to understand a wide range of inputs means that LLMs are not restricted to a narrow band of predefined command patterns. It allows LLMs to understand a broad set of actions and interpret expressive adjectives to modify the content being created. LLMs are architecturally very flexible and are capable of being integrated into software systems in multiple ways. An LLM can make an API call (known as "tool use") based on user input, can interact with an external service (via the model context protocol, or MCP), or can directly interact with its own internal model of the artifact being generated. This gives LLMs flexibility in how they can take action based on a wide range of commands, thereby executing a range of procedural generation tasks. Further, LLMs can also describe the artifacts they are creating, answer questions about them, and provide ideas for future directions to explore. In every capacity, LLMs are more flexible than the previous generation of script-based natural language interfaces.

The broad expressiveness and capability of LLMs creates the potential to use them as a natural language front-end for procedural generation tasks. Indeed, their flexibility allows them to be considered more broadly for game design tasks, as well as procedural generation of levels [2]. LLMs support a conversational interaction style where the human designer can make multiple requests in a row and seek feedback from the LLM as part of an interactive session with a procedural generator. In this way, LLMs support a mixed-initiative style of interaction with a procedural generator, in addition to a more traditional command-driven generation approach.

Many issues arise when seeking to use LLMs for procedural generation, including: what might a designer plausibly want to *say* to a generator? How are game worlds *represented* by and for the LLM? How does an LLM *manipulate* a game world? How well does an LLM actually *understand and reason* about a game world? We explore these issues in the remainder of this paper. In Section 3 we give an overview of different kinds of statements a designer might want to make to a LLM during procedural design tasks, such as generation requests of varying granularity. Section 4 describes ways of representing a game world both internally and externally to an LLM. We introduce the *world facts database*, an external representation approach that describes the component elements of a level in a manner suitable for use in LLM prompting. Section 5 explores tradeoffs among different kinds of software architectures for manipulating game worlds. For example, direct API calls via tool use can provide greater accuracy, but limit the range of potential actions. Finally, in Section 6 we describe the creation of a QA dataset, TinyTownQA, that can support fine-tuning of LLMs. This dataset is used to characterize the ability of text-based and multi-modal (vision) LLMs to understand and answer questions about tile-based game worlds. Such understanding is a necessary foundation for LLMs to support procedural generation and design feedback tasks for 2D tile-based worlds.

2 Background

Use of natural language interfaces for mixed-initiative procedural design systems traces its roots back to the 1970s. Yona Friedman gives an early vision in the introduction to Chapter 3 of Negroponte's *Soft Architecture Machines* (1976), viewing human-machine collaboration as a kind of dialog between the human designer and the object being designed, with the computer acting as a translator of design acts into their impacts and implications on the object [17]. Bolt's "Put That There" (1980) system added natural language interfaces into the mix, and explored the combined use of natural language and gesture inputs to take action in a 2D virtual world projected on a screen [3]. The system supported actions such as the ability to create and move objects, and to identify objects by pointing.

Prior to the advent of LLMs, work on natural language interfaces with a focus on procedural generation was sparse, with Mobramaein's work on natural language interfaces for Pong game variants (2018) [13] and wave generation (2020) [12] among the few examples.

Since 2023, use of natural language prompts for LLMs has been explored for the creation of levels for a roguelike brawler game called Metavoidal [16], Sokoban [21], Super Mario World [18], and Angry Birds-like games [1][20]. The Word2World system uses LLMs

to create a narrative for a game world and then populate it with internally consistent characters and items [15]. The game traversal benchmark is used to explore LLMs ability to navigate through 2D tile maps [14]. The Five-Dollar Model creates game maps by converting a natural language input (using a sentence transformer) into a vector that then feeds a custom upsampling convolution network, generating tile-based game maps [11]. In work that is similar to ours, the Quick Custom Map Generation (QMBS) system takes a text prompt as an input and uses an LLM to generate hexagonal tile based game worlds [6].

LLMs have been used to generate entire games by asking them to output the rules of the game using a game description language. Hu et al. explore the a variety of different prompts for the creation of a maze game, where an LLM outputs the rules in VGDL [8]. In a similar vein, Tanaka and Simo-Serra use an LLM to generate games by outputting GDL [19]. The DreamGarden system offers a more ambitious approach, using an LLM as a planner to refine an intial natural language game description into a functional game running in Unreal [4].

More broadly, research involving applications of LLMs to games has grown rapidly over the past three years, with several surveys providing an overview of the research landscape. Gallotta et al. [7] and Yang et al. [22] survey all applications of LLMs in games, while Maleki and Zhao survey techniques for procedural content generation, including an examination of uses of LLMs [5] and Mao et al. survey uses of generative AI (including LLMs) for procedural generation in games [10].

3 Communicating with the Game World

LLMs make it possible to have conversational interactions with a game world. What are some of the things human designers would like to be able to say to a generator during a design session?

Creation requests at multiple levels of granularity. We would like to request the generator to modify the game world for us. Broad generation requests involve actions that affect all or large portions of a game world, such as creating an entire level. This is a large granularity generation request. A medium granularity request would involve modifying an existing level to add a large feature without recreating the entire level. For example, adding an island into a pre-existing lake in a game map. Fine-grained generation requests involve adding or modifying single items in a game world (e.g., adding a mushroom to a forest), or repetitions of such actions. A variation on this theme is the use of prompts which apply only to selected sub-regions, an idea explored in Reframer [9].

Identifying items and objects. A challenge in any conversational system is how to correctly identify locations and items in the game world. This is necessary to describe which object is being acted upon in update and delete actions, or to describe the location where a create action will take place. While it is always possible to use exact coordinates, this is tedious, especially compared to using a mouse pointer. Relative positioning is one approach often used in conversation where a location is given relative to another object. For example, one can make a request to "add a house to the left of the forest". More complex relative positions are area identifiers, such as asking for mushrooms to be placed *within* a forest, or placing buckets *around* a well. A challenge in this approach is uniquely identifying elements. In the previous example, if there were two distinct clusters of trees in a game level, it is ambiguous which forest is intended. There should ideally be ways to name items in the game world for further use in the conversation, or, alternatively, the LLM should be capable of detecting such ambiguity and ask the designer to disambiguate.

Retrieval/query requests. A conversational system should be able to answer questions about the game world. Queries can be about object quantities ("how many keys are in the map") or qualities ("how many windows are in the brown house"). Ideally the system could answer more complex queries, such as the number of tiles an object uses ("how many tiles does the brown house have") or the distance between two objects.

Use of emotionally descriptive keywords. Ideally, a conversational system should able to guide generation towards a particular emotional response in the player, such as "Create a dark, foreboding forest". Mobramaein explored the use of emotion keywords for wave generation (e.g., an angry sea) in [12].

4 Representing the Game World

In order to perform actions that modify a game world, an LLM needs to have a representation of that game world. This representation needs to support the following activities:

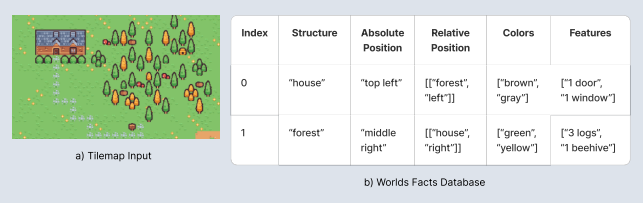
- *Reflection.* The LLM needs to understand what items are currently in the game world (e.g., houses, castles, trees, paths, signs, collectibles). It should understand their properties (e.g., a house has two windows and one door) and how much space each item uses in the game world.
- *Identification.* The LLM needs to be able to translate a natural language statement identifying an item in the game world to the item itself.
- *Positioning.* For items in the game world, the LLM needs to understand their absolute ("the house in the center of the level") and relative positions ("the well to the right of the house").
- *Analysis.* Ideally, an LLM should be able to provide design feedback to a human designer based on its understanding of the game world and the desired goals for that world. This requires an understanding of the items present in the world, and their relative positions.

4.1 Internal and External Representations

Broadly, there are two approaches to providing LLMs with game world context. An *internal* representation uses the LLM itself to maintain a coherent model of the game world in the conversational context. In this approach, user prompting directly modifies the model's "understanding" of the game world by affecting the conversation history. An *external* representation maintains a model of the game world in a format managed outside of the LLM. In this case, before any request by the user is processed, the representation is provided to the LLM (e.g., a tile map, or an image of a level), and the model either outputs the updated tilemap, or a set of actions required to update the tilemap. Examples of external representations for a 2D tilemap include a 2D array of tile identifiers with associated key (e.g., 1, 1, 4, 25, ... where 1=grass, 4=flower, 25=house), a 2D array of characters where each character represents a type of tile

(e.g., ggfh... where g=grass, f=grass with flower, h=house), a 2D array of strings (e.g., "grass", "grass", "grass+flower", "house"), or, for vision models, a bitmap image of the level itself (e.g., a png or jpg). The QMBS system uses Chinese characters to represent each tile using a single character, taking advantage of the fact that each character is a whole word [6].

Anecdotally, our experience has been that character- and word-based approaches work better than using tile IDs, even with the provision of a key. However, this performance does vary across different LLMs, suggesting there are model-to-model variations in training on grid-based representations (and the utility of fine-tuning to improve understanding of grid worlds). We hypothesize this is due to LLMs having a greater understanding of word co-occurrence frequencies than number co-occurrence.



a) Tilemap Input

Index	Structure	Absolute Position	Relative Position	Colors	Features
0	"house"	"top left"	[["forest", "left"]]	["brown", "gray"]	["1 door", "1 window"]
1	"forest"	"middle right"	[["house", "right"]]	["green", "yellow"]	["3 logs", "1 beehive"]

b) Worlds Facts Database

Figure 1: a) A sample tilemap representing a tile-based game world, and b) a portion of the associate world facts database with metadata about the tilemap's items.

4.2 World Facts Database

A variation on the external approach is to provide the LLM with a direct representation of the game world along with a dictionary of key-value pairs that provide metadata about it. We call this dictionary a *world facts database*. The world facts database includes a list of items in the game world, descriptive text about each item, and the relative position of the items with each another (see Figure 1). In its role as a store of world information used by other artificial intelligence components in a generation system, the world facts database acts like a traditional knowledgebase or blackboard. The world facts database is intended to be provided to the LLM as part of the prompt that includes the user request. This world fact information is intended to improve the LLMs ability to understand and take accurate action in the game world. In this way, it acts similar to a retrieval augmented generation (RAG) type system, where relevant context is supplied to the LLM as part of a request in order to increase response accuracy.

One challenge is maintaining consistency between the tile-based representation and the world facts representation of a level. We developed a sample algorithm for extracting a world facts database from 2D tilemaps created using the Kenney "Tiny Town" asset pack, a series of 16x16 pixel 2D tiles that can be composed to make scenes comprised of a grassy background, with grey or brown houses, castles, forests, pathways, fenced-in areas, and various decoration items (beehive, well, wheelbarrow, etc.).[1] We use this tileset as a running example due to its simplicity (it has a relatively limited number of tiles) and ability to create a large number of interesting top-down 2D levels. The input to the algorithm is a 2D array of tile identifiers representing the visible 2D tilemap. The algorithm is provided with an array of structure identification information, including the name of each structure type (e.g. "house"), features that may appear on it (e.g. "door," "window"), and the tile IDs associated with each part of the structure. A flood-fill algorithm is employed to locate contiguous tiles of the same structure type in the tilemap. Each identified structure then undergoes descriptive analysis:

- *Positional description (absolute).* The structure's location on the map is classified into predefined spatial categories (e.g., top-left, top-center, top-right, middle-left, center, middle-right, etc.) in relation to the overall map.
- *Positional description (relative).* The structure's location is classified into predefined spatial categories in relation to other structures on the map (e.g. "...to the top left of the forest").
- *Substructure analysis.* Any smaller components within the primary structure (e.g. roofs for houses), usually multi-tile.
- *Feature identification.* Key features within each structure (e.g., windows and doors for houses, mushrooms and beehives in forests), usually one tile in size.
- *Color characteristics.* The dominant colors of the structure.

The analyzed structure is then stored as an object in an array, encapsulating the parameters structure type, unique ID, bounding box coordinates, and the descriptive attributes generated through this process. This dataset can subsequently be serialized into a string format, providing a natural language description of the input tilemap, suitable for inclusion in LLM prompts.

5 Manipulating the Game World

There are two broad architectural approaches for using an LLM to support conversational interactions with a game world. In the *tool-based* approach, an LLM is instructed to call a user-supplied function that most closely matches the human designer's natural language request (see Figure 2). The LLM identifies the correct function, and extracts parameters for the function call. This approach has the advantage of precise semantics: once the correct function is identified, the code implementation of the function ensures consistent execution of the desired action, reducing the likelihood and impact of LLM hallucination. However, it has the drawback of only supporting a fixed range of actions, reducing the expressive advantages of LLMs. Further, some LLMs find it challenging to creating composite actions involving a series of multiple tool use calls.

In the *direct manipulation* approach, the LLM directly acts upon a representation of the level to perform any action requested by the human designer (see Figure 3). No external code functions are called to perform requested actions, the LLM manipulates the game world without any intermediary code. The key advantage of this approach is expressivity: it takes full advantage of the LLM to interpret and execute a wide range of natural language statements, and supports a much broader range of conversational interaction patterns. The primary drawback is that LLMs struggle to correctly interpret and understand tile-based worlds, especially to understand composite elements such as a house comprised of multiple tiles, or elements with built-in constraints, such as a path that must connect two houses.

[1] https://kenney.nl/assets/tiny-town

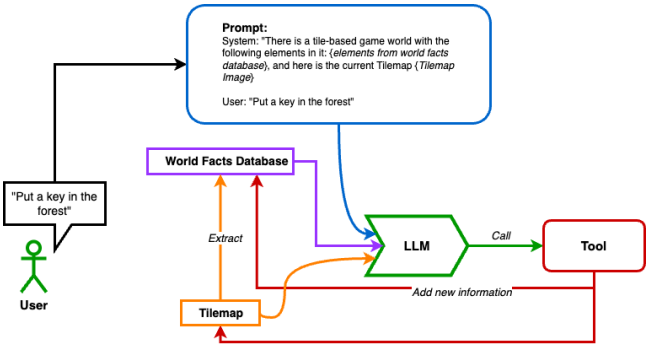


Figure 2: Diagram demonstrating the workflow for a Tool-based system

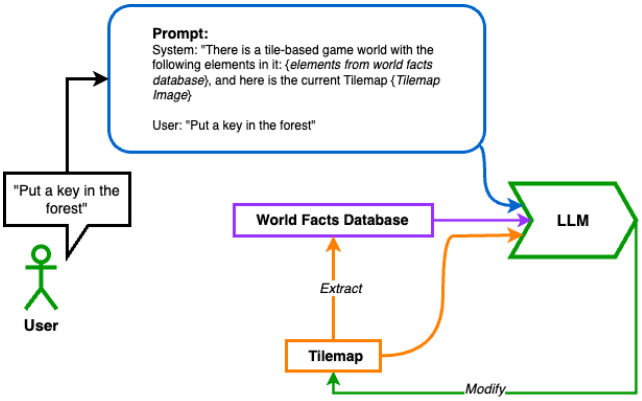


Figure 3: Diagram demonstrating the workflow for a Direct Manipulation system

We implemented two prototypes to explore these approaches. Our experience with the direct manipulation approach was disappointing because the LLM found it challenging to understand the tile-based world and hence this limited its ability to accurately perform human designer requests. We do not report on this experience further in this section, but note that these challenges drove our desire to characterize the performance of LLMs for game world understanding tasks in Section 6. All prototypes were implemented in JavaScript using the Phaser game development environment and employed the Tiny Town tile set.

Prototype 1. This prototype used the Gemini Flash 1.5 model via Langchain using tool calling (see Figure 4).[2] Available functions can place a single tile, remove a tile, describe the tile at a location, and create an entire tilemap. This uses the *internal* representation of the game world, where working in multiple sessions requires re-prompting the LLM to refresh its context. The world representation is a list of tiles, where each tile has an (x,y) coordinate and a descriptive label limited to a set of 7 options ("dirt", "grass", "flower_grass", "bush", "tree", "mushroom", "wheelbarrow"). This prototype was used to explore how to identify items and positions in the game world. It successfully interpreted cardinal directions ("place a tree in the southeast"), relative directions ("place a tree above the mushroom"), and screen space directions ("place a tree in the top center") from only a system prompt. Gemini seemed to be able to remember contexts based on the chat history, especially when coordinates were specified by the tools so that they could be referenced in the future by the LLM. Gemini seemed to only be able to handle about 5 iterations when being told to batch add any sprite to the world. For example, when asked to make a forest it will only add about 5 trees.

Prototype 2. This prototype used the ChatGPT 3.5 model via the OpenAI REST API using tool calling.[3] Available tool use functions are "placeItemAdjacentToTree", "placeItemAdjacentToPath", "placeItemsInsideFencedAreas" and "placeItemAnywhere". No world model was presented to the LLM, though it did have the conversation history and a list of available objects that could appear in the game world. This led to the prompts needing to be in a fairly specific syntax for the LLM to correctly output useful data. Cardinal and screen space directions had to be hard coded for the LLM to recognize them. The LLM was successful at figuring out which function to call after the input is parsed, for example if the user typed "Generate ..." or "Please place ...", it understood to call the function to place items on the map. After parsing the user input, the LLM works well with directional wording such as, "Top" or "North" when describing where to place the objects. The LLM was able to batch add large amounts of objects from a single call.

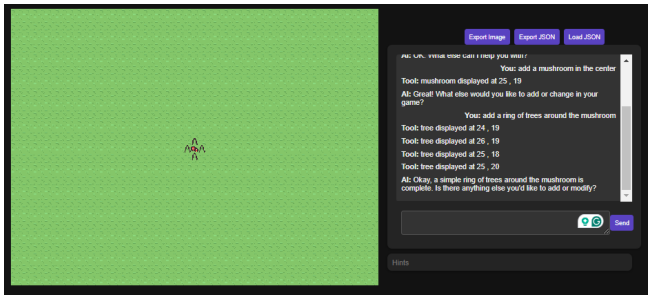


Figure 4: UI for Prototype 1, Gemini Flash 1.5 with tool use.

6 Understanding the Game World

In order to improve the ability of LLMs to understand their contents and take accurate action in game worlds, we wished to evaluate their performance answering questions about tile-based worlds.

6.1 TinyTownQA Dataset

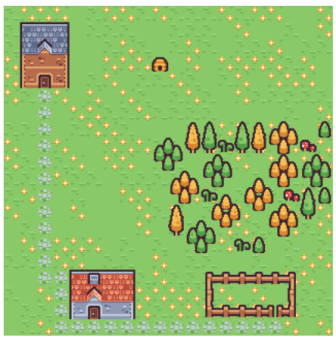
To perform this analysis, we needed a dataset of example tile-based worlds with associated descriptive information. We create such a database by first creating a procedural generator capable of generating Tiny Town game worlds, along with their associated world facts database.[4]

The tilemap generator begins by recursively dividing the map into sections of varying sizes using a space-partitioning method, ensuring that each section falls within defined size constraints.

[2] https://github.com/SentientDragon5/CMPM-118-LLM-Test

[3] https://github.com/GigzPumpking/z3demo

[4] https://github.com/collectioncard/Tiny-Town-Dataset-Generator



World Facts

A closed fenced-in area, with fences at (12, 16) with width 6 and height 3.
Forest 0 at (9, 7) with width 11 and height 8.
House 0 is a gray House with 0 windows at (4, 16) with width 4 and height 3.
A beehive at (9, 3).
House 1 is a brown House with 0 windows at (1, 1) with width 3 and height 4.
Path connecting points: (16, 18) (5, 18) (2, 4)

Relationships:

Relationships for Closed Fence at (12, 16):
- is below Forest 0 at (9, 7) (distance: 9.49 tiles).
- is to the right of House 0 at (4, 16) (distance: 8.00 tiles).
- is diagonally below and to the right of House 1 at (1, 1) (distance: 18.60 tiles).

Relationships for Forest 0 at (9, 7):
- is above Closed Fence at (12, 16) (distance: 9.49 tiles).
- is diagonally above and to the right of House 0 at (4, 16) (distance: 10.30 tiles).
- is diagonally below and to the right of House 1 at (1, 1) (distance: 10.00 tiles).

Relationships for House 0 at (4, 16):
- is to the left of Closed Fence at (12, 16) (distance: 8.00 tiles).
- is diagonally below and to the left of Forest 0 at (9, 7) (distance: 10.30 tiles).
- is diagonally below and to the right of House 1 at (1, 1) (distance: 15.30 tiles).

Relationships for House 1 at (1, 1):
- is diagonally above and to the left of Closed Fence at (12, 16) (distance: 18.60 tiles).
- is diagonally above and to the left of Forest 0 at (9, 7) (distance: 10.00 tiles).
- is diagonally above and to the left of House 0 at (4, 16) (distance: 15.30 tiles).

Sample Questions

Question	Answer	Distractors		
Which object is located directly to the left of the Closed Fence?	House 0	Forest 0	House 1	Beehive
What is the distance between the Closed Fence and House 1?	18.60 tiles	9.49 tiles	10.30 tiles	8.00 tiles
How many windows does House 0 have?	0 windows	2 windows	3 windows	1 window
Which object is directly below Forest 0?	Closed Fence	House 0	House 1	Beehive
What is the width of the Forest 0?	11 tiles	8 tiles	9 tiles	10 tiles
Identify the coordinates of House 1.	(1, 1)	(4, 16)	(9, 3)	(12, 16)

Figure 5: Tilemap, associated world facts, and sample multiple choice questions from the TinyTownQA dataset.

Houses are designed with walls, doors, roofs, and windows, while forests contain a mix of single-tile and multi-tile trees, bushes, and mushrooms. Fences can appear as fully enclosed areas, L-shaped enclosures, or straight-line barriers, contributing to the spatial diversity of the town. Decorative objects, such as logs, wheelbarrows, and beehives, are placed randomly within designated sections. Each section is then assigned a random generation function that determines whether it remains empty or contains decorative objects, houses, fences, or forests. This independent generation process allows for procedural variation and ensures that each section adheres to specific structural rules. Each section is first generated on a local tile grid and then integrated into the global map.

Once all sections are generated, the pathfinding system connects the house doors and the fence gates to create a road network. This is accomplished by identifying critical endpoints and applying A* pathfinding to compute the shortest routes between them. Kruskal's algorithm is then used to construct a minimum spanning tree to prevent redundant pathways. The generator constructs a world facts database that records the exact coordinates, spatial dimensions, and relationships between objects (e.g., "The house is adjacent to the fenced area").

Once a sample tilemap and associated world facts database has been generated, they are provided to an LLM (GPT4o-mini) that creates at least 20 multiple choice questions per map using this information. During question generation, the LLM only had access to the World Facts Database and did not have access to the images. The LLM was requested to create questions on one of the following topics: relative spatial relationships, absolute distances between objects, properties of objects (e.g., which house has one window), and how to identify objects on the map. The LLM was also given the option of creating other types of questions. Each multiple choice question had 1 correct answer and 3 distractor answers. The resulting dataset contained questions with incorrect answers. A manual cleanup step was performed where each question was human-verified for correctness, with incorrect questions removed from the dataset.

The final dataset is comprised of 31 unique maps with associated world facts data, and a total of 642 vetted multiple choice questions. An example entry is shown in Figure 5. The dataset is published on HuggingFace as TinyTownQA.[5]

6.2 World Reasoning Evaluation

We performed experiments with varying world representations: world facts database (Facts), tilemap image (Images), an array of tile identifiers (Array), and combinations of these (World Facts + Images, Array + Images) in order to evaluate the performance of LLMs when performing world understanding tasks. In the first ("Facts"), an LLM is provided only with the world facts database (example in Figure 5) for a level before being asked to answer a question. The second experiment ("Facts + Images") provides the image of the tilemap level and the world facts data, to assess whether the LLM will improve its performance if it has two complementary sources of information about the level. The third experiment provides just the rendered tilemap level image, with no world facts data, to evaluate whether the image has information content similar to the world facts database ("Images"). Because questions often refer to structures as "House 2" or "Forest 0", the images were supplemented with a text-based mapping of feature names to their top left coordinate. In the last experiment, the LLM was provided only with a base tilemap array (an array of integer tile identifiers) and the locations of each landmark, such as "Forest0 at (11,5)" ("Array"). The same experiment was performed again with a rendered level Image as well ("Array + Images").

In each experiment, the set of questions was the same (TinyTownQA) and contained 642 unique questions across 31 unique maps. The questions were asked separately with no context other than the prompt and the singular question, so that the questions and answers did not have the context of each other. In all experiments, the LLMs are "off-the-shelf" and have not been fine-tuned with the TinyTownQA map data or world facts.[6]

The results of evaluating the TinyTownQA dataset against multiple LLMs are shown in Table 1. Overall, GPT-4o (world facts + images) performed the best of all models, but the accuracy is similar to that of Deepseek R1 and Gemini-Flash-2.0. Notably, all but the

[5]https://huggingface.co/datasets/collectioncard/TinyTownQA
[6]https://github.com/collectioncard/LLM-QA-Analysis

Table 1: TinyTownQA Evaluation Results

Model Name	World Representation	Accuracy
Qwen2.5-1.5B	Facts	0.517
DeepSeek-R1-Distill-Qwen-1.5B	Facts	0.586
Qwen2.5-7B	Facts	0.785
Gemini-flash-2.0	Facts	0.941
Gemini-flash-2.0	Images	0.642
Gemini-flash-2.0	Array	0.669
Gemini-flash-2.0	Array + Images	0.673
GPT-4o-mini	Images	0.514
GPT-4o-mini	Facts	0.911
GPT-4o-mini	Facts + Images	0.908
GPT-4o	Facts	0.944
GPT-4o	Facts + Images	0.972
Deepseek R1	Facts	0.956

largest models tested were unable to use the extra context provided by including tilemap images. The image data typically acts as a distractor, reducing performance, instead serving to ground model output. Testing image data alone resulted in poor performance, suggesting that models are not capable of reasoning about image-represented levels. Additionally, providing models with tile arrays seems to result in similar performance to that of just images. In general, larger models performed much better than smaller models, with a difference of over 40 points between our highest and lowest scoring run. A notable exception to this is the Google Gemini flash 2.0 model, which was able to keep up with the higher performing models despite being the smallest Gemini 2.0 model available. Finally, with Gemini flash 2.0, the performance using the tilemap image, array of tile ids, or the ids+image was all broadly similar.

7 Conclusions

Our experience from creating two prototype conversational procedural generation systems shows that the tool use architecture supports generation and retrieval tasks by focusing the range of actions that can be taken, and providing dedicated implementations for those actions. However, tool use has the drawback of limiting the range of potential actions and expression supported by the LLM. The direct manipulation architecture offers the promise of being more expressive, but requires an LLM be capable of directly interacting with the game world representation. We explore the current ability of LLMs to reason about game world information via the creation of a QA dataset based on procedurally generated maps and world facts based on the Tiny Town tileset. This evaluation shows that large parameter models are capable of highly accurate reasoning about game world information and should be capable of supporting rich conversational interactions. Current models are not capable of reasoning with level image data or raw tilemap representations, an area for further exploration.

References

[1] Febri Abdullah, Pittawat Taveekitworachai, Mury F. Dewantoro, Ruck Thawonmas, Julian Togelius, and Jochen Renz. 2024. The First ChatGPT4PCG Competition. *IEEE Transactions on Games* 16, 4 (2024), 971–980. doi:10.1109/TG.2024.3376429

[2] Asad Anjum, Yuting Li, Noelle Law, M Charity, and Julian Togelius. 2024. The Ink Splotch Effect: A Case Study on ChatGPT as a Co-Creative Game Designer. In *Proceedings of the 19th International Conference on the Foundations of Digital Games* (Worcester, MA, USA) *(FDG '24)*. Association for Computing Machinery, New York, NY, USA, Article 18, 15 pages. doi:10.1145/3649921.3650010

[3] Richard A Bolt. 1980. "Put-that-there" Voice and gesture at the graphics interface. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*. 262–270.

[4] Sam Earle, Samyak Parajuli, and Andrzej Banburski-Fahey. 2024. DreamGarden: A Designer Assistant for Growing Games from a Single Prompt. arXiv:2410.01791 [cs.HC] https://arxiv.org/abs/2410.01791

[5] Mahdi Farrokhi Maleki and Richard Zhao. 2024. Procedural Content Generation in Games: A Survey with Insights on Emerging LLM Integration. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 20, 1 (Nov. 2024), 167–178. doi:10.1609/aiide.v20i1.31877

[6] Binlin Feng, MingYang Su, Keyi Zeng, and Xiu Li. 2024. Player-Oriented Procedural Generation: Producing Desired Game Content by Natural Language. In *HCI in Games*, Xiaowen Fang (Ed.). Springer Nature Switzerland, Cham, 260–274.

[7] Roberto Gallotta, Graham Todd, Marvin Zammit, Sam Earle, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. 2024. Large Language Models and Games: A Survey and Roadmap. *IEEE Transactions on Games* (2024), 1–18. doi:10.1109/tg.2024.3461510

[8] Chengpeng Hu, Yunlong Zhao, and Jialin Liu. 2024. Game Generation via Large Language Models. arXiv:2404.08706 [cs.AI] https://arxiv.org/abs/2404.08706

[9] Tomas Lawton, Francisco J Ibarrola, Dan Ventura, and Kazjon Grace. 2023. Drawing with Reframer: Emergence and Control in Co-Creative AI. In *Proceedings of the 28th International Conference on Intelligent User Interfaces* (Sydney, NSW, Australia) *(IUI '23)*. Association for Computing Machinery, New York, NY, USA, 264–277. doi:10.1145/3581641.3584095

[10] Xinyu Mao, Wanli Yu, Kazunori D Yamada, and Michael R. Zielewski. 2024. Procedural Content Generation via Generative Artificial Intelligence. arXiv:2407.09013 [cs.AI] https://arxiv.org/abs/2407.09013

[11] Timothy Merino, Roman Negri, Dipika Rajesh, M Charity, and Julian Togelius. 2023. The Five-Dollar Model: Generating Game Maps and Sprites from Sentence Embeddings. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 19, 1 (Oct. 2023), 107–115. doi:10.1609/aiide.v19i1.27506

[12] Afshin Mobramaein. 2020. *Natural Language Interfaces for Procedural Content Generation in Games*. PhD Thesis. University of California, Santa Cruz.

[13] Afshin Mobramaein, Morteza Behrooz, and Jim Whitehead. 2018. CADI—A Conversational Assistive Design Interface for Discovering Pong Variants. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 14, 1 (Sep. 2018), 194–200. doi:10.1609/aiide.v14i1.13042

[14] Muhammad Umair Nasir, Steven James, and Julian Togelius. 2024. GameTraversalBenchmark: Evaluating Planning Abilities Of Large Language Models Through Traversing 2D Game Maps. arXiv:2410.07765 [cs.CL] https://arxiv.org/abs/2410.07765

[15] Muhammad U. Nasir, Steven James, and Julian Togelius. 2024. Word2World: Generating Stories and Worlds through Large Language Models. arXiv:2405.06686 [cs.CL] https://arxiv.org/abs/2405.06686

[16] Muhammad U Nasir and Julian Togelius. 2023. Practical PCG Through Large Language Models. In *2023 IEEE Conference on Games (CoG)*. 1–4. doi:10.1109/CoG57401.2023.10333197

[17] Nicholas Negroponte. 1976. *Soft Architecture Machines*. The MIT Press. doi:10.7551/mitpress/6317.001.0001

[18] Shyam Sudhakaran, Miguel González-Duque, Matthias Freiberger, Claire Glanois, Elias Najarro, and Sebastian Risi. 2023. MarioGPT: Open-Ended Text2Level Generation through Large Language Models. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 54213–54227. https://proceedings.neurips.cc/paper_files/paper/2023/file/a9bbeb2858dfbdbd4c19814e5d80ec60-Paper-Conference.pdf

[19] Tsunehiko Tanaka and Edgar Simo-Serra. 2024. Grammar-based Game Description Generation using Large Language Models. *IEEE Transactions on Games* (2024), 1–14. doi:10.1109/TG.2024.3520214

[20] Pittawat Taveekitworachai, Febri Abdullah, Mury F. Dewantoro, Yi Xia, Pratch Suntichaikul, Ruck Thawonmas, Julian Togelius, and Jochen Renz. 2024. ChatGPT4PCG 2 Competition: Prompt Engineering for Science Birds Level Generation. arXiv:2403.02610 [cs.AI] https://arxiv.org/abs/2403.02610

[21] Graham Todd, Sam Earle, Muhammad Umair Nasir, Michael Cerny Green, and Julian Togelius. 2023. Level Generation Through Large Language Models. In *Proceedings of the 18th International Conference on the Foundations of Digital Games* (Lisbon, Portugal) *(FDG '23)*. Association for Computing Machinery, New York, NY, USA, Article 70, 8 pages. doi:10.1145/3582437.3587211

[22] Daijin Yang, Erica Kleinman, and Casper Harteveld. 2024. GPT for Games: An Updated Scoping Review (2020-2024). arXiv:2411.00308 [cs.AI] https://arxiv.org/abs/2411.00308