Consistent Game Content Creation via Function Calling for Large Language Models

Roberto Gallotta
Institute of Digital Games
University of Malta
Msida, Malta
roberto.gallotta@um.edu.mt

Antonios Liapis
Institute of Digital Games
University of Malta
Msida, Malta
antonios.liapis@um.edu.mt

Georgios Yannakakis
Institute of Digital Games
University of Malta
Msida, Malta
georgios.yannakakis@um.edu.mt

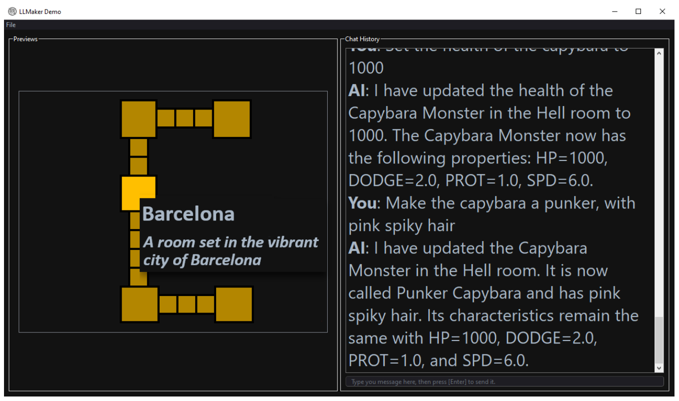
***Abstract*—Tools for designing content require a medium that allows the designer to efficiently express their creativity, and a system that ensures the content being designed adheres to the domain of interest. Interacting with Large Language Models (LLMs) via natural language is extremely intuitive for a human designer, although it remains largely unexplored. However, this approach has a limitation: LLMs are prone to hallucinations and they tend to ignore parts of the user request in their responses. One workaround is to let LLM use tools such as *function calling* to ensure consistency of the content. We formalise this approach by proposing *LLMaker*, a general framework for consistent video game content generation empowered by LLMs, bridging the gap between creative vision and technical execution. We demonstrate *LLMaker*'s application in generating dungeon crawler level layouts, comparing it against alternative LLM-based methods for content generation over multiple tests, testing for consistency of the outputs and elapsed time per request.**

I. Introduction

The design of video games, in their entirety or piecemeal, has been a focal point of research in procedural content generation (PCG) [1]. Different approaches have been explored, some relying on human feedback to guide the search process, such as in mixed-initiative co-creation [2]. However, practical applications often suffer from cluttered user interfaces, inundating designers with numerous controls and options, rendering the tool more difficult to master.

Large language models (LLMs), recently popularised by OpenAI with models like GPT-3 [3] are trained on vast text corpora spanning diverse fields, acquiring an illusion of reasoning, at times appearing human-like in their responses [3]. Text generation via LLMs starts with an initial text prompt, which usually defines some rules for the model, an optional conversation history and the user input. Without additional domain knowledge, responses rely solely on user input or training data. This is the *zero shot* paradigm. Adding domain knowledge through examples enables *few-shot* reasoning [4]. While a simple improvement, learning by examples can lead to wrong assumptions by the model. *Chain-of-thought* [5] simulates the reasoning behind the decision making process,

This project has received funding from the European Union's Horizon 2020 programme under grant agreement No 951911.


LLMaker Demo
File
Previews
Chat History
Barcelona
A room set in the vibrant city of Barcelona
Type you message here, then press [Enter] to send it.

1000
AI: I have updated the health of the Capybara Monster in the Hell room to 1000. The Capybara Monster now has the following properties: HP=1000, DODGE=2.0, PROT=1.0, SPD=6.0.
You: Make the capybara a punker, with pink spiky hair
AI: I have updated the Capybara Monster in the Hell room. It is now called Punker Capybara and has pink spiky hair. Its characteristics remain the same with HP=1000, DODGE=2.0, PROT=1.0, and SPD=6.0.

Fig. 1: A screenshot of our chat-based level design interface, *LLMaker*, on test case T5. On the left pane, the generated level layout, with rooms (larger squares) and corridors (smaller squares). On the right pane, the chat area with the conversation between designer and LLM.

increasing the accuracy in responses given by the LLM on a variety of tasks. As a way to further combat hallucinations, finetuned models that supported *function calling* [6] were introduced: in this case, the LLM relies on a separate system or API to obtain data for their responses. Following [7], we identify *context inconsistency*, where parts of the context provided to the LLM is ignored, as the class of hallucinations we are most concerned with, and evaluate our approach against it by defining our own model accuracy measure.

Automated content generation in video games is a matter of computational creativity [8]. Procedural content generation (PCG) aims to create functional and enjoyable game elements. Content generators for video games can operate with minimal user input [9], or involve a human designer in a mixed-initiative [2] framework, allowing designers to interact with automated systems to quickly iterate over content design while respecting designer agency and authorship, rapidly assisting in the generation of levels [10], in-game content [11], and game mechanics [12]. LLMs, democratised via APIs or open source models, have spurred research into their potential for generating video game content, yielding mixed results [13]. In this work, LLMs are implemented to streamline user interaction by making the back-end system *transparent*, reducing the need for designers to manage every detail of the domain during the design process. Our own preliminary experiments in this domain suggest structured content such as JSON is effective for generating valid game content. However, it can fall short when requests become complex or we attempt to refine the artefact over multiple iterations, leading to errors, poorly formatted data, and violations of domain constraints. We define this as a lack of *content consistency*. LLMs follow user instructions, which can include domain knowledge, improving the quality of the responses compared to their zero-shot reasoning performance [14]. In the context of content generation, function calling can be used as a way for the LLM to modify the artefact following the designer's intention, with no risk of illegal changes and without including the entire artefact in the generated response. This approach offers two main benefits: maintaining valid artefacts and swift response times.

Inspired by these hypotheses, we present *LLMaker*, a framework for co-designing video game content (see Figure 1). In *LLMaker*, language models serve as expert intermediaries, translating human designer requests into *formal*, *valid* inputs for content generation. This ensures consistency and adherence to domain constraints. Unlike existing LLM-powered PCG applications [15], [16], *LLMaker* allows for *iterative refinement* and *fine-grained modifications* of a game level and its content in a "chat-like" environment.

II. LLMaker

LLMaker is a chat-only tool facilitating iterative game content co-design. It serves as a bridge between human designers and a content generation system, utilising LLMs to translate designer requests into instructions for a content generation system. Unlike traditional tools, *LLMaker* enables designers to express creativity freely via natural language, while ensuring the content adheres to domain rules. This is achieved by validating content via a back-end system. *LLMaker* can then create content for different domains, requiring just different back-ends to interact with. This versatility streamlines the design process, making it less cumbersome for users. *LLMaker* is, to our knowledge, the first design tool powered by LLMs that enables designers to work under this paradigm.

We implement *LLMaker* in a simple graphical user interface Windows application utilising OpenAI's GPT-3.5-turbo-0613 [3] via the developer API. Upon launch, it initiates an empty level for users to refine through natural language input. Users can request modifications or clarifications, and the system responds accordingly. While the system can call functions as needed, it is not always mandatory. Users can freely explore content properties and gather feedback through conversation with the LLM as conversational agent [17] before requesting any change to the level.

LLMaker relies on a predefined set of functions to operate. During design the LLM can provide a function name and the necessary arguments to call it with. These are generated based on context and domain constraints. For instance, if asked to add an enemy, the system may assign a random speed within pre-specified ranges. Users can also provide specific values, although the system ensures they adhere to constraints. The outcome of function execution is relayed back to the user, either as a success message or an explanation of any failures, following the paradigm of functional errors [18].

TABLE I: User requests for test case T5. Each request is submitted sequentially via *LLMaker*.

Create 3 rooms, each connected to the next one, all set in a different European city
Add a goblin archer in the first room
Also add two zombies
Now generate a room connected to the first one, set in underground Atlantis
Put a couple of evil mermaids in Atlantis
Place multiple ocean-themed traps in the corridor to Atlantis
Place a single treasure chest in all rooms, each containing a piece of a treasure map
Remove the chest containing the second piece of the treasure map
Add another room connected to Atlantis, set in Hell
Place two fallen angels armed with flaming swords
Change one of the angels to a capybara monster
Set the health of the capybara to 1000
Make the capybara a punker, with pink spiky hair

A. Domain: Dungeon Crawler Levels

In this study, we replicate the popular RPG video game *Darkest Dungeon*[1]. The game immerses players in a Lovecraftian world, where they navigate underground mazes while managing stress levels of their hero squad. To emulate this environment realistically, we employ a context-aware generative grammar, ensuring parameters define the properties of each element within the level.

The level generation process begins with a root node, leading to the expansion of rooms and corridors, each housing encounters such as enemies, traps, or treasure chests. Enemies are characterised by combat statistics and species, while traps and treasure chests offer distinct challenges and rewards. Unlike Darkest Dungeon, here each room and each entity has to be uniquely identified by its name. Valid levels are levels that comply to this grammar, and satisfy all pre-specified constraints.

III. Experimental Protocol

As introduced in Section I, we aim to validate two hypotheses: (1): *function calling results in more valid artefacts being generated*; and (2): *interaction with function calling is faster compared to other methods*. We test this by comparing function calling configuration against different prompting techniques on five separate test cases (T1-T5) available in this project repository[2]. We report one test case in Table I. These test cases simulate realistic scenarios for designing a game level, varying in content complexity, specificity of user requests, and different design control flow.

[1]Red Hook Studios, 2016

[2]We release the source code for this project at https://github.com/gallorob/llmaker_functioncalling

We run each test case 10 times, with seed randomisation. We compare four different ways to prompt the LLM: Zero-shot, Few-shots [14], Chain-of-Thought [5], and Function Calling [19]. The latter provides descriptions of functions and corresponding arguments in the prompt[3]. Each interaction includes the system prompt and a JSON representation of the current level. We do not include the history of the conversation (i.e., past interactions between user and system), as it is not required to fulfil the user requests. While function calling directly results in the content being modified, the other approaches generate the updated level as JSON, which we attempt to parse to validate changes.

We define four possible outcomes for each interaction: *Parser Fail* when the output is not parseable (i.e.: a malformed JSON, such as missing properties in an entity), *Domain Fail* when the level is invalid (i.e.: the level grammar is not respected, such as a missing corridor between two rooms), *Design Fail* when the output doesn't match user requests, and finally *Success* when the output is valid and fulfils the requests. Function calling always produces valid levels due to back-end constraints but may misinterpret or fail to meet all user requests (leading to *Design Fails*).

We monitor changes to the level during execution for easier debugging. We keep track of how many *responses* are returned by the LLM before failure, and the *time elapsed for each response* to be generated. We test whether content matches user expectations, ensuring both objective (e.g., adding a new room) and subjective (e.g., adding "a couple" of enemies should result in more than one new enemy, and less than the maximum number of enemies) requests are met. On failures, we terminate the test case, tracking the triggering request and ignoring the failing response elapsed time. We expect varying average response times per test case, as different techniques require a different number of tokens being exchanged to and from the LLM.

IV. Results

Table II presents the results of 10 runs with random seed variation per test case (T1-T5). Other prompting techniques often fail quickly, usually within 1 or 2 responses. Baseline methods never complete a test case without failing. Function calling never triggers parser or domain fails, though it may cause design fails. It also achieves the highest average responses per test case with lowest per-request elapsed time. We find that parser failures occur when the language model disregards the prompt to always generate a JSON response, instead asking the user for further details. Domain grammar violations are the primary cause of failures in baseline models. Most commonly, connecting corridors fail to be created when generating a new room, despite this being specified in the prompt. Another issue arises when the model deviates from the level grammar, such as defining loot in a treasure chest incorrectly.

[3]We report all prompts in https://github.com/gallorob/llmaker_functioncalling/tree/main/prompts

TABLE II: Results for different prompting methods on all test cases averaged from 10 independent runs. Fails measure the number of instances of 10 runs that failed, while Responses and Time (per Request) are averaged from 10 runs and include the 95% Confidence Interval. Responses and Time values with $\star$ indicate significantly outperforming all other configurations on this Test Case.

Prompting	Test Case	Fails ↓			Responses ↑	Time (s) ↓
		Pars.	Dom.	Des.		
Zero Shot	T1	0	10	0	3 ± 0.0	6.1 ± 0.1
	T2	0	10	0	3 ± 0.0	10.2 ± 0.2
	T3	0	10	0	2 ± 0.0	7.6 ± 0.3
	T4	10	0	0	1.1 ± 0.2	2.8 ± 1.5
	T5	7	3	0	3 ± 0.0	26.6 ± 0.3
Few Shot	T1	0	10	0	4 ± 0.0	15.6 ± 0.8
	T2	0	10	0	4 ± 0.0	12.7 ± 0.2
	T3	0	10	0	7 ± 0.0	21.4 ± 0.2
	T4	0	10	0	3 ± 0.0	5.9 ± 0.1
	T5	0	10	0	3 ± 0.0	26.9 ± 2.6
Chain of Thought	T1	0	10	0	3 ± 0.0	16.4 ± 0.4
	T2	0	10	0	3 ± 0.7	13.7 ± 3.2
	T3	0	10	0	2 ± 0.0	11.9 ± 0.3
	T4	0	10	0	3 ± 0.0	12.8 ± 0.2
	T5	10	0	0	1.5 ± 0.9	67.3 ± 4.8
Function Calling	T1	0	0	7	$7\pm0.0^\star$	9.6 ± 0.6
	T2	0	0	0	$9\pm0.0^\star$	$6.9\pm0.4^\star$
	T3	0	0	0	$10\pm0.0^\star$	$5.7\pm0.0^\star$
	T4	0	0	1	$11\pm0.0^\star$	4.9 ± 0.1
	T5	0	0	0	$13\pm0.0^\star$	$8.5\pm0.1^\star$

In terms of response time, a Wilcoxon signed-rank test with Bonferroni correction for multiple comparisons showed that function calling was significantly faster ($p < 0.05$) than all other prompting methods in 4 of 5 test cases. Baseline models' response time increases with level complexity due to the need to regenerate the JSON level description. Chain-of-Thought has the longest response time in the most complex test case (T5), and generally its response time is between 100% (T1) and 600% (T5) slower than function calling. This is because Chain-of-Thought not only generates the JSON level description but also the decision-making process for altering the level. Other baseline methods also show slower response times compared to function calling, with Zero-Shot and Few-Shot being 200% (T5) and 300% (T3) slower at worst, respectively. The response time for function calling remains consistent regardless of level complexity as it only generates function parameters and a brief summary text response. However, in one instance (T4), Zero-Shot achieves a lower elapsed time than function calling, while failing to provide the required JSON structure.

V. Discussion

The results in Section IV show that *LLMaker*'s function calling approach can still trigger design fails. It also maintains consistent response speed regardless of level complexity. Among baselines, Chain-of-Thought performs the worst due to longer response times. We find that our approach validates both hypotheses presented in Section III: function calling always generates consistent video game content with a response time which would be acceptable for real-time interaction (under 10 seconds).

There are however limitations. The first one is potential bias in test cases selection, as these were compiled by the authors. Secondly, while our goal was to test the different prompting techniques on realistic prompts, their difficulty is subjective. A more detailed and larger set of test cases is left for future work. Thirdly, the design choices for the functions used to generate and modify the game level and the domain constraints have to be defined by the programmer, and the designer cannot alter either of these, limiting their creative freedom. Finally, GPT-3.5-Turbo-0613 was chosen as it was found to perform well on a multitude of tasks [20], however different models such as GPT-4 may yield much different results.

Despite the effectiveness of GPT models, their proprietary nature raises concerns. Future iterations of *LLMaker* may explore open-source alternatives such as Nexusraven 13B [21]. While this work emphasises content consistency, future research should consider usability (i.e.: how useful the LLM responses were to a human designer) and aesthetics, leveraging foundation models such as Stable Diffusion [22] to bring the descriptions provided by the LLM to life. Additionally, *LLMaker* lacks proactive assistance, a known issue being actively researched [23]. Also, completion criteria are absent, risking unplayable or imbalanced levels. Adding constraint checks would ensure playability and balance.

Finally, we note that *LLMaker* is a mixed-initiative design assistant that is in constant dialogue with the designer, opening up future work for mixed-initiative tools where interaction between human and machine is based exclusively on natural language. However, this warrants evaluation for cognitive demand and usability through studies focusing on natural language dialogue.

VI. Conclusions

In this paper we introduced *LLMaker*, an innovative tool for co-creative video game content design empowered by large language models. In *LLMaker*, the interaction between designer and system is entirely based on natural language, with the LLM translating user queries into properly formatted requests to a back-end system via function calling. We demonstrated that function calling is superior to other LLM-based methods for generating content in terms of prompt adherence and domain constraints satisfaction. Additionally, *LLMaker* consistently processes user requests in a few seconds, serving the user with the updated content almost in real-time. Overall, *LLMaker* demonstrates how function calling for LLMs can be efficiently implemented in a game design tool. With this work, we hope to inspire other researchers to explore the paradigm of co-creativity based on natural language, its strengths, limitations, and possibilities.

References

[1] A. Liapis, "10 years of the PCG workshop: Past and future trends," in *Proc. of the Foundations of Digital Games International Conf.*, 2020.

[2] G. N. Yannakakis, A. Liapis, and C. Alexopoulos, "Mixed-initiative co-creativity," in *Proc. of the Foundations of Digital Games International Conf.*, 2014.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Proc. of the 34th International Conf. on Neural Information Processing Systems*, 2020.

[4] S. Yu, J. Liu, J. Yang, C. Xiong, P. Bennett, J. Gao, and Z. Liu, "Few-shot generative conversational query rewriting," in *Proc. of the International ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2020.

[5] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Proc. of the International Conf. on Neural Information Processing Systems*, 2023.

[6] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," in *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[7] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," *arXiv preprint arXiv:2311.05232*, 2023.

[8] A. Liapis, G. N. Yannakakis, and J. Togelius, "Computational game creativity," in *Proc. of the Innovative Computing and Cloud Computing International Conf.*, 2014.

[9] M. Cook and S. Colton, "Ludus ex machina: Building a 3D game designer that competes alongside humans," in *Proc. of the Innovative Computing and Cloud Computing International Conf.*, 2014.

[10] E. Butler, A. M. Smith, Y.-E. Liu, and Z. Popovic, "A mixed-initiative tool for designing level progressions in games," in *Proc. of the ACM Symposium on User Interface Software and Technology*, 2013.

[11] A. Summerville and M. Mateas, "Mystical tutor: A magic: The gathering design assistant via denoising sequence-to-sequence learning," in *Proc. of the AAAI Conf. on Artificial Intelligence and Interactive Digital Entertainment*, 2021.

[12] R. van Rozen, "A pattern-based game mechanics design assistant," in *Proc. of the Foundations of Digital Games International Conf.*, 2015.

[13] R. Gallotta, G. Todd, M. Zammit, S. Earle, A. Liapis, J. Togelius, and G. N. Yannakakis, "Large language models and games: A survey and roadmap," *arXiv preprint arXiv:2402.18659*, 2024.

[14] A. Madotto, Z. Liu, Z. Lin, and P. Fung, "Language models as few-shot learner for task-oriented dialogue systems," *arXiv preprint arXiv:2008.06239*, 2020.

[15] C. Hu, Y. Zhao, and J. Liu, "Game generation via large language models," *arXiv preprint arXiv:2404.08706*, 2024.

[16] V. Kumaran, D. Carpenter, J. Rowe, B. Mott, and J. Lester, "Procedural level generation in educational games from natural language instruction," *IEEE Transactions on Games*, 2024.

[17] M. Wahde and M. Virgolin, *Conversational Agents: Theory and Applications*, ch. 12. World Scientific, 2022.

[18] E. Buoannno, "Functional error handling," in *Functional Programming in C#*, ch. 6, Manning Publications Co., 2017.

[19] H. Jiang, L. Ge, Y. Gao, J. Wang, and R. Song, "Large language model for causal decision making," *arXiv preprint arXiv:2312.17122*, 2023.

[20] X. Chen, J. Ye, C. Zu, N. Xu, R. Zheng, M. Peng, J. Zhou, T. Gui, Q. Zhang, and X. Huang, "How robust is GPT-3.5 to predecessors? a comprehensive study on language understanding tasks," *arXiv preprint arXiv:2303.00293*, 2023.

[21] Nexusflow.ai team, "NexusRaven-V2: surpassing GPT-4 for zero-shot function calling." https://nexusflow.ai/blogs/ravenv2, 2023. Accessed 27 Feb 2024.

[22] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2022.

[23] Z. Lin, U. Ehsan, R. Agarwal, S. Dani, V. Vashishth, and M. Riedl, "Beyond prompts: Exploring the design space of mixed-initiative co-creativity systems," in *Proc. of the International Conf. on Computational Creativity*, 2023.