Fill-in-the-Blank Microgames

Akash Surendran
Northeastern University
Boston, MA, USA
surendran.a@northeastern.edu

Seth Cooper
Northeastern University
Boston, MA, USA
seth.cooper@gmail.com

Abstract

In this work we explore an approach for creating personalized microgames, drawing inspiration from fill-in-the-blank stories such as Mad Libs. In fill-in-the-blank stories, the reader is given a list of word categories such as place, number, or adjective, and provides keywords to fill in the list. These keywords are then copied into the blanks of a short story, with often humorous results. We develop a similar process for generating microgames, where the keywords are used to fill in the blanks of a Large Language Model game generation prompt. Rather than extending the capabilities of LLMs for game generation, part of our aim is using LLMs as-is for generating unpolished, small-scale, personalized microgames that can be played for a brief duration. We compare two template prompting strategies, apply these across four genres, and evaluate the generated games based on incorporation of keywords, variety, and qualitative observations.

Keywords

procedural content generation, game generation, generative AI

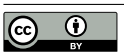
ACM Reference Format:
Akash Surendran and Seth Cooper. 2026. Fill-in-the-Blank Microgames. In *Foundations of Digital Games (FDG '26), August 10–13, 2026, Copenhagen, Denmark.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3815598.3815713

1 Introduction

There has long been interest in reducing barriers to game development through new tools and languages [10, 11, 18, 19, 26, 36, 38], empowering players to create their own games personally relevant to themselves [20]. However, game development remains a challenging undertaking.

One existing approach to allowing novices to create their own *short stories* is fill-in-the-blank stories such as Mad Libs [31]. In fill-in-the-blank stories, the reader is given a list of word categories such as place, number, or adjective; they then provide keywords to fill in the list. These keywords are copied over into the blanks of a template short story, oftentimes with humorous results due to the incongruence of the various keywords and the story template.

Using categorized keywords to fill a template may also lend itself well to the generation of short, personalized *microgames*. Microgames sometimes exist in collections of social party games such as WarioWare [23]. Such an application has previously been a motivating application of *automated game design* [21].



FDG '26, Copenhagen, Denmark

ACM ISBN 979-8-4007-2495-4/26/08
https://doi.org/10.1145/3815598.3815713

One recent approach that has the potential to support this process is applying Large Language Models (LLMs) to generating code for microgames. In particular, the capabilities of LLMs may not currently handle fully creating the scale and complexity of an industry-standard game. Rather than extending the capabilities of LLMs to larger games, we instead focus on the types of games that small language models, or current large models (such as LLMs or diffusion models) may be able to generate: microgames, which are only be played for a short time, and thus issues such as lack of polish or coherence might be overlooked and potentially even humorous.

We take an approach inspired by fill-in-the-blank stories, using prompt templates containing fill-in-the-blanks for specific categories of words: nouns, adjectives, numbers, etc. The player personalizes a template by providing concrete keywords for each of the blanks in the template, which is then given to an LLM to create the code for the game.

In this approach, we developed a technique for iteratively constructing detailed game prompt templates designed to be robust to different keywords, and compared these to simple baseline game prompt templates that simply list the game elements as keywords. We developed and used these templates across four genres, generating a total of ninety-six games from two instantiations of each genre and evaluating them for e.g. playability, noticeable presence of keywords, and uniqueness.

We found that both the detailed and simple templates' games yielded similar results, although the simple prompts took more care in including the specified keywords in a far more literal sense. The simple prompts also appeared to have more variety in the generated games. We also qualitatively analyzed properties of the generated games.

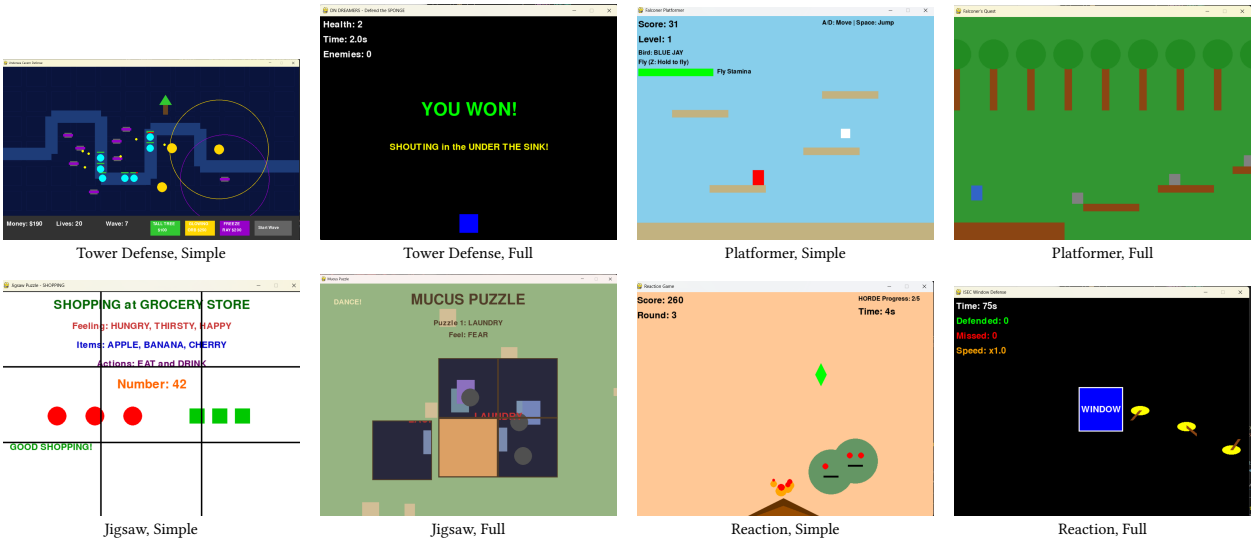
This work contributes an exploration of fill-in-the-blank generation of microgames and prompting strategies using LLM-based code generation.

2 Related Work

This work is related to a number of areas of research that look at generating games or code.

Within the realm of procedural content generation [27], generating entire games has received interest using "classical" techniques. Some early work, such as METAGAME [25] and Ludi [4] focused on generating boardgames. Other early work looked at WarioWare-like microgames, supported by existing knowledge bases, to generate game mechanics capturing relationships between entities [21, 35]. Domain specific languages such as VGDL have been used to evolve games [22]. Other work in game generation has explored it as a co-evolution or search process partly guided by a human designer [7, 8].

With the possibility of generating content, personalization of content by generating or adapting content has also been explored.



Tower Defense, Simple | Tower Defense, Full | Platformer, Simple | Platformer, Full

Jigsaw, Simple | Jigsaw, Full | Reaction, Simple | Reaction, Full

Figure 1: One simple and one full game from each genre.

Dynamic difficulty adjustment has been applied in a range of games [13, 17], though games can be personalized for other player experiences [39, 40]. This has included searching or evolving to generate content toward specific player experience [28, 29], or training reinforcement learning agents to generate personalized content [3, 30]. Some recent work has also explored using "generative AI" to create personalized card games [20].

More recently, generative AI based on large language models has been explored for generating code [6, 9]. This has also led to applications of generative AI different aspects of game development [12, 32], such as generating code in game description languages [16, 34] or level generation [33], or even entire Unity game prototypes [15].

Another recently explored approach to generating games, or interactive environments, is *world models* [14]. World models for the most part may be considered interactive movies that are generated at the pixel level frame-by-frame and incorporate user input.

Such interactive diffusion models could be used as game engines [37]. While initial world models could be trained for specific applications, more recent approaches such as Genie [5, 24] can be more general and incorporate an initial prompt to generate the environment. However, these approaches can often only be played for a short time before ending, and may also lack common game elements such as a goal or puzzles.

3 Approach

The overall goal of this experiment was to explore supporting the creation of personalized microgames in a simplified and approachable way for those with little-to-no experience designing games. The approach we explored was using an LLM to generate code for the games. For a game development environment, we used the Pygame library in Python [1]. The games were code-based and did not incorporate additional assets, and thus only code using Pygame was necessary to be generated. The model used for this experiment was Claude Sonnet 4.5 [2]. The web version was used in the earlier stages of template development, while the API was called in the Python code for the actual generation of the games analyzed in the experiment. This model was used during the months of July 2025 through November 2025.

For a particular type of game to be generated (e.g. in a specific genre), a *prompt template* is created. The prompt template has blanks associated with different categories of word (e.g. noun, verb, number, etc), to be filled in by keywords (e.g. cat, run, 67, etc). These mappings from category to keyword are provided by the user separately from the template. The current system implementation takes the category to keyword mapping as a JSON file, which we filled out manually, and used to complete the prompt template.

When developing the templates, the approach stemmed from multiple previous development sessions with Claude, in which the user was heavily involved in the creation and debugging process. A simple pattern was noticed — more detailed and specific prompts commonly yielded more complex games and fewer bugs to fix.

The initial approach after this discovery was to make the initial prompt given to Claude as detailed and intricate as possible, containing information about the gameplay, the objectives, specific game values (such as health), and more. The goal was to simplify user-AI interaction by reducing the chat duration required to make a functional game. Next, it was observed that having all this game information in one place is similar to what a game design document is used for. Thus, the approach became to create a simple design document for the desired game and feed it into Claude, rather than trying to fit as much information as possible into a single message.

As the goals were to simplify interactions as well as make simple game development approachable, requiring the user to think critically about game systems and user experience seemed rather complex from a beginner's standpoint. Due to this, the idea was developed to create design document templates that corresponded to a certain genre of game, such as a platformer or a jigsaw puzzle game. The template would outline the core gameplay and detail common bugs to avoid, while input spaces were left throughout

Genre	TD	Platform	Reaction	Jigsaw	Overall (%)
Playability (game counts)					
Runs?	12 : 12	12 : 12	12 : 12	12 : 11	100 : 98
Playable?	12 : 12	12 : 12	11 : 9	12 : 11	98 : 92
Completable?	12 : 6	5 : 3	4 : 8	3 : 6	50 : 48
Fun?	0 : 5	3 : 3	7 : 5	5 : 4	31 : 35
Keywords (average keyword counts per game - rounded)					
Possible	13	17	10	12	–
Title	1 : 1	1 : 1	1 : 1	1 : 1	8 : 8
Text	3 : 5	4 : 7	4 : 8	3 : 8	28 : 57
Other	1 : 2	2 : 6	3 : 4	2 : 1	17 : 25
Unique	5 : 6	6 : 11	5 : 8	5 : 8	41 : 64
Game distinctness (game counts)					
Distinct	4 : 5	8 : 11	10 : 11	7 : 9	60 : 75

Table 1: Values for different measures given as comparisons between Full : Simple. Overall percentages for game counts are out of the 96 games. Overall percentages for keywords are average percentage of keywords used averaged across the four genres.

each template for the user to decide game themes and designs as well as game values, such as how many levels are in the game or how long the game would last. This "fill-in-the-blank" style of development is what resulted from an attempt to make this process more fun for the user, hiding the details of the template from the user and giving them just the JSON file with only the categories and asking them to input the keywords.

The input spaces in the template documents were formatted as `{{WORD}}`, The `WORD` in the brackets would correspond to a key in the JSON files. For example, if a template prompt contains `It should give the feeling of {{VERB_1}} in a {{PLACE}}.`, then the key-value pairs in the JSON file `VERB_1: "running"` and `PLACE: "house"` would become `It should give the feeling of running in a house.` When the program runs, the entire chosen template document gets converted into a string and any instances are replaced. After doing this for all key-value pairs, the entire string would get sent to the API.

4 Experiment and Analysis

To explore the kinds of games generated by this approach, and the impact of different kinds of prompt templates on the generated games, we generated games using both the more extensive "design document templates" described above, referred to as the *full* templates, and straightforward *simple* templates that contained only the genre of game desired, the keywords found in the JSON, and a directive to include the keywords in the game. The addition of a simplified prompt allowed us to test whether more detailed, robust prompts yielded better generated games.

We looked at 4 genres of game (and thus 4 templates of each type, full and basic): Tower Defense, Platformer, Reaction Timing, and Jigsaw Puzzle.

The experiment began with both authors creating 8 JSON files each, split equally between the 4 template design documents. Each author created 2 JSON files per genre.

The program was given all of the 8 JSON files created by a single author — 8 game ideas that each had 2 versions generated. One of these versions was generated using the full template and one with the simple template. Additionally, the program created 3 variants of each version to observe differences between games generated with the exact same prompt. This came to a total of 48 games generated from the 8 JSON files constructed by a single author. These 48 games were then given to the other author for analysis, aiming to reduce bias from prior expectations of the keywords. The genre and template type were also hidden during analysis of playability and keyword usage. The content of the prompt templates themselves was hidden from one of the authors, although the other author developed the templates and was familiar with them. Each author thus analyzed the 48 games from the other author, resulting in a total of 96 games that were analyzed and from which data was collected. Some screenshots of the generated games are shown in Figure 1.

The games were analyzed through multiple lenses. First, the authors analyzed the overall playability through four questions, each building on the last. The first question was simply if the game ran. Perhaps it would be due to bugs in the code or some other issue, but if the game cannot run, the other questions cannot be asked, as the game cannot be analyzed further. Some games required minor cleanup such as deleting extraneous text at the end of the code, which still counted as running. The second question was if the game is playable. This was a test to see if the player could reasonably interact with the game through input of some kind, such as keyboard or mouse. The third question was if the game was completable. This meant the game would need to have a defined state where the game was declared to be over, and in which the player could achieve a winning state or adjacent result, such as a high score in a never-ending game such as Tetris. The fourth and final question was if the game is fun. This question is the most subjective part of this experiment and was mostly left up to the personal tastes of the author analyzing a game.

The next lens through which the games were analyzed were the usage of the keywords included in the JSON files used to generate the games. This was, once again, analyzed through four categories. These were: how many keywords were used in the title, how many keywords were used in textual elements, how many keywords were used in any other elements, and finally how many unique keywords were used in the entire game. This was done to analyze how well Claude could include specific elements of the prompt. Although this was somewhat subjective, we aimed to consistently count keywords when clearly apparent in the game, such as in text, numbers, or images appearing.

We also kept rough notes on observations when playing the game for later discussion and synthesis.

The final lens of analysis, which was game distinctness — evaluating how many games had a great enough degree of uniqueness to be considered a unique game and not simply a variant of another. To be unique from one another, games had to be noticeably distinct in either gameplay or visual design. For example, two tower defense games made from the same simplified prompt may have a similar path that enemies follow and similar units to purchase and place, but one becomes distinct from the other by having enemy speed increase with each wave and displaying the attacking range of each placed unit. The experience is enhanced, and is therefore distinct. This uniqueness analysis was done by the first author only looking

at groups of six games for each individual JSON file, although while knowing the genre and template type.

Templates used for this work and analysis of the games is available at https://osf.io/yr73h/.

Data: A summary of the data is given in Table 1. In terms of playability, both template types are similar, suggesting that more robust prompts do not necessarily yield better games. The simple templates appear to have an advantage in number of keywords incorporated into the game and the distinctness of the games generated.

This disparity in keyword usage is best seen in the Jigsaw Puzzle games. The games with the full prompt featured puzzles that depicted the actual objects or things specified in the JSON files, while the focus of the puzzles from the simple games was to cram as many keywords as possible, always resulting in the puzzle being a panel with all the keywords written on it rather than an image of any sort.

General Observations: After playing and discussing the games, the authors noted some patterns in the generated games. These are preliminary and were synthesized from informal discussions between the authors rather than a full formal analysis.

In terms of *number keywords*, we found that it was important to stay in reasonable ranges for what they represent (e.g. time, collectibles), which could be difficult when filling in a blank disconnected from its context. Sometimes, numbers appeared in a game but seemed disconnected from their actual use; for example, a number on screen might imply that there are five coins to collect in a level, when in fact there are only two.

Additionally, games could be challenging due to *ambiguity* in the generated game elements. For example, sometimes puzzles would have multiple pieces that were just blank spaces, requiring the player to blindly guess where they might be placed. If the number of blank spaces was high enough, a puzzle game would be deemed not completable due to the number of combinations.

Expanding on this, we noticed that while the vast majority of games had a system of controls that allowed them to be playable, nearly half of all games either made it impossible or unreasonably difficult to reach a completed state, usually from a lack of *tuning and balance*. Examples include the aforementioned blank spaces, reaction games spawning points to be clicked at a rate faster than the authors could react, and platformer games containing platforms that are too high to be jumped on.

Often keywords would be incorporated using what we termed *shape pictures*: pictures of the keyword assembled using geometric shapes and colors. For example, glasses, tomatoes, or stars. In some cases it appeared that just colors were used on simple rectangles; for example blue water, yellow lard, or green mucus.

To some extent, the generated games may have captured keyword *emotion*. Primarily, anger could be associated with rapidly moving red objects.

Unsurprisingly, we found that keywords being on the screen as text helped to *reinforce* a sense that they had been incorporated (even if they were incorporated in some other way — e.g. a shape picture, emotion, or counter). Simply having the keyword text on-screen may be a straightforward strategy to make the game personalization clear.

5 Discussion & Conclusion

In this experiment, we explored the concept of fill-in-the-blank microgames using LLMs, and compared games generated with detailed and simplified prompts. We learned that games generated in either manner showed generally comparable results, with slight variation based on the genre of game generated, suggesting that one method is not necessarily superior to the other, and simple prompts may work well enough for microgames.

There are a few limitations with this experiment. First and foremost, only Claude Sonnet 4.5 was used in the generation of games. A single model is not representative of the overall capabilities of all generative AI models accessible to the general public, let alone those with more exclusive accessibility and more capabilities. It is vital to compare these results across multiple LLM models to gauge the average capabilities of the available AI generation tools. Moving forward it may be worth it to explore, and produce template documents that contain far greater detail to determine if is capable of scaling its design accordingly. We are also interested in applying world models to generate interactive games directly rather that using LLM code generators.

Another limitation is the informal nature of the analysis performed by the authors. While we attempted to reduce bias in the analysis, both authors are experienced in game design and development, and are related to the development of the experiment itself. An experiment using an unrelated, random selection of people to negate bias as much as possible was not conducted. This also means the only people analyzing the games were the two authors, and each game could only be analyzed by one author. Although not as subjective as determining whether a game was fun, the other questions asked to determine playability may show inconsistent results based on how the authors determined their answers.

While the observations made during this experiment support the idea that a moderately detailed prompt may end up limiting the capabilities of a generative AI model, it does not imply that a massively detailed and specific prompt would yield better results.

There are other potential concerns to consider. We hope that approaches such as we have presented open up new avenues where games can be applied and who can create them, rather than taking the place of existing game development approaches. We also hope that this approach will be amenable to smaller models, specifically for microgames, that can be run locally. There is also the possibility to generate offensive content, which should be taken into consideration.

Finally, player studies would help establish what players think of the games, for example if they find them enjoyable or humorous.

Acknowledgments

This work was partially carried out during a co-op in the Khoury College of Computer Sciences.

References

[1] [n. d.]. Pygame. https://www.pygame.org/news.
[2] Anthropic. [n. d.]. Claude. https://claude.ai/.
[3] Colan F. Biemer and Seth Cooper. 2022. Level Assembly as a Markov Decision Process. In *Proceedings of the Experimental AI in Games Workshop.*
[4] Cameron Browne and Frederic Maire. 2010. Evolutionary Game Design. *IEEE Transactions on Computational Intelligence and AI in Games* 2, 1 (March 2010), 1–16. doi:10.1109/TCIAIG.2010.2041928

[5] Jake Bruce, Michael D. Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Maria Elisabeth Bechtle, Feryal Behbahani, Stephanie C. Y. Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando de Freitas, Satinder Singh, and Tim Rocktäschel. 2024. Genie: Generative Interactive Environments. In *Forty-First International Conference on Machine Learning*.

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. https://arxiv.org/abs/2107.03374v2.

[7] Michael Cook. 2022. Puck: A Slow and Personal Automated Game Designer. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 18, 1 (Oct. 2022), 232–239. doi:10.1609/aiide.v18i1.21968

[8] Michael Cook, Simon Colton, and Jeremy Gow. 2017. The ANGELINA Videogame Design System—Part I. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 2 (June 2017), 192–203. doi:10.1109/TCIAIG.2016.2520256

[9] Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. 2024. Evaluating Large Language Models in Class-Level Code Generation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*. Association for Computing Machinery, New York, NY, USA, 1–13. doi:10.1145/3597503.3639219

[10] Tamara Duplantis, Isaac Karth, Max Kreminski, Adam M. Smith, and Michael Mateas. 2021. A Genre-Specific Game Description Language for Game Boy RPGs. In *2021 IEEE Conference on Games (CoG)*. 1–8. doi:10.1109/CoG52621.2021.9619109

[11] Kaylah Facey, Cynthia Li, Chris Martens, and Seth Cooper. 2025. Game Behaviour Trees Using Tile Rewrite Rules. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 21, 1 (Nov. 2025), 216–226. doi:10.1609/aiide.v21i1.36825

[12] 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

[13] Miguel González-Duque, Rasmus Berg Palm, David Ha, and Sebastian Risi. 2020. Finding Game Levels with the Right Difficulty in a Few Trials through Intelligent Trial-and-Error. *arXiv:2005.07677 [cs]* (June 2020). arXiv:2005.07677 [cs]

[14] David Ha and Jürgen Schmidhuber. 2018. World Models. (March 2018). doi:10.5281/zenodo.1207631 arXiv:1803.10122 [cs]

[15] Amna Hassan. 2025. Automated Unity Game Template Generation from GDDs via NLP and Multi-Modal LLMs. doi:10.48550/arXiv.2509.08847 arXiv:2509.08847 [cs]

[16] Chengpeng Hu, Yunlong Zhao, and Jialin Liu. 2024. Game Generation via Large Language Models. In *2024 IEEE Conference on Games (CoG)*. 1–4. doi:10.1109/CoG60054.2024.10645597

[17] Robin Hunicke. 2005. The Case for Dynamic Difficulty Adjustment in Games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE '05)*. Association for Computing Machinery, New York, NY, USA, 429–433. doi:10.1145/1178477.1178573

[18] Chris Klimas. 2009. Twine.

[19] Max Kreminski, Melanie Dickinson, Joseph Osborn, Adam Summerville, Michael Mateas, and Noah Wardrip-Fruin. 2020. Germinate: A Mixed-Initiative Casual Creator for Rhetorical Games. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 16, 1 (Oct. 2020), 102–108. doi:10.1609/aiide.v16i1.7417

[20] Brandon Lyman, Ala Ebrahimi, James Cox, Szeyi Chan, Christopher Barney, and Bob De Schutter. 2024. Cardistry: Exploring a GPT Model Workflow as an Adapted Method of Gaminiscing. In *Proceedings of the 19th International Conference on the Foundations of Digital Games (FDG '24)*. Association for Computing Machinery, New York, NY, USA, 1–4. doi:10.1145/3649921.3656984

[21] Mark J. Nelson and Michael Mateas. 2007. Towards Automated Game Design. In *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*, Roberto Basili and Maria Teresa Pazienza (Eds.). Springer, Berlin, Heidelberg, 626–637. doi:10.1007/978-3-540-74782-6_54

[22] Thorbjørn S. Nielsen, Gabriella A. B. Barros, Julian Togelius, and Mark J. Nelson. 2015. Towards Generating Arcade Game Rules with VGDL. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. 185–192. doi:10.1109/CIG.2015.7317941

[23] Nintendo. 2003. WarioWare, Inc.: Mega Microgame$!

[24] Jack Parker-Holder and Shlomi Fruchter. 2025. Genie 3: A New Frontier for World Models. https://deepmind.google/blog/genie-3-a-new-frontier-for-world-models/.

[25] Barney Pell. 1992. *METAGAME in Symmetric Chess-like Games*. Technical Report UCAM-CL-TR-277. University of Cambridge, Computer Laboratory. doi:10.48456/tr-277

[26] Alexander Repenning. 2017. Moving beyond Syntax: Lessons from 20 Years of Blocks Programing in AgentSheets. *Journal of Visual Languages and Sentient Systems* 3 (July 2017), 68–91. doi:10.18293/VLSS2017-010

[27] Noor Shaker, Julian Togelius, and Mark J. Nelson. 2016. *Procedural Content Generation in Games*. Springer.

[28] Noor Shaker, Georgios Yannakakis, and Julian Togelius. 2010. Towards Automatic Personalized Content Generation for Platform Games. In *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'10)*. AAAI Press, Stanford, California, USA, 63–68.

[29] Noor Shaker, Georgios N. Yannakakis, Julian Togelius, Miguel Nicolau, and Michael O'Neill. 2012. Evolving Personalized Content for Super Mario Bros Using Grammatical Evolution. In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'12)*. AAAI Press, Stanford, California, USA, 75–80.

[30] Tianye Shu, Jialin Liu, and Georgios N. Yannakakis. 2021. Experience-Driven PCG via Reinforcement Learning: A Super Mario Bros Study. In *2021 IEEE Conference on Games (CoG)*. IEEE Press, Copenhagen, Denmark, 1–9. doi:10.1109/CoG52621.2021.9619124

[31] Leonard Stern and Roger Price. 1958. Mad Libs. https://madlibs.com/.

[32] Alexandru Ternar, Alena Denisova, João M. Cunha, Annakaisa Kultima, and Christian Guckelsberger. 2025. Generative AI in Game Development: A Qualitative Research Synthesis. doi:10.48550/arXiv.2509.11898 arXiv:2509.11898 [cs]

[33] 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 (FDG '23)*. Association for Computing Machinery, New York, NY, USA, 1–8. doi:10.1145/3582437.3587211

[34] Graham Todd, Alexander G. Padula, Matthew Stephenson, Éric Piette, Dennis J.N.J. Soemers, and Julian Togelius. 2024. GAVEL: Generating Games via Evolution and Language Models. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 110723–110745.

[35] Mike Treanor, Bryan Blackford, Michael Mateas, and Ian Bogost. 2012. Game-O-Matic: Generating Videogames That Represent Ideas. In *Proceedings of the The Third Workshop on Procedural Content Generation in Games (PCG'12)*. Association for Computing Machinery, New York, NY, USA, 1–8. doi:10.1145/2538528.2538537

[36] V Buckenham. 2024. Downpour. https://downpour.games.

[37] Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. 2025. Diffusion Models Are Real-Time Game Engines. doi:10.48550/arXiv.2408.14837 arXiv:2408.14837 [cs]

[38] Christina Volioti, Vasileios Martsis, Apostolos Ampatzoglou, Euclid Keramopoulos, and Alexander Chatzigeorgiou. 2025. Codeless3D: Design and Usability Evaluation of a Low-Code Tool for 3-D Game Generation. *IEEE Transactions on Games* 17, 2 (June 2025), 296–307. doi:10.1109/TG.2024.3424894

[39] Georgios N. Yannakakis and Julian Togelius. 2011. Experience-Driven Procedural Content Generation. *IEEE Transactions on Affective Computing* 2, 3 (2011), 147–161.

[40] Jichen Zhu and Santiago Ontañón. 2020. Player-Centered AI for Automatic Game Personalization: Open Problems. In *Proceedings of the 15th International Conference on the Foundations of Digital Games (FDG '20)*. Association for Computing Machinery, New York, NY, USA, 1–8. doi:10.1145/3402942.3402951