You Have Thirteen Hours in Which to Solve the Labyrinth: Enhancing AI Game Masters with Function Calling

Jaewoo Song, Andrew Zhu, Chris Callison-Burch
University of Pennsylvania
{jwsong05,andrz,ccb}@seas.upenn.edu

Abstract

Developing a consistent and reliable AI game master for text-based games is a challenging task due to the limitations of large language models (LLMs) and the complexity of the game master's role. This paper presents a novel approach to enhance AI game masters by leveraging function calling in the context of the table-top role-playing game "Jim Henson's Labyrinth: The Adventure Game." Our methodology involves integrating game-specific controls through functions, which we show improves the narrative quality and state update consistency of the AI game master. The experimental results, based on human evaluations and unit tests, demonstrate the effectiveness of our approach in enhancing gameplay experience and maintaining coherence with the game state. This work contributes to the advancement of game AI and interactive storytelling, offering insights into the design of more engaging and consistent AI-driven game masters.

1 Introduction

Imagine a world where the power of storytelling meets the ingenuity of artificial intelligence, giving rise to game masters that can weave captivating narratives and adapt to the players' choices in real-time. This is the vision that drives our research into enhancing AI game masters for table-top role-playing games (TTRPGs). However, the realization of this vision is hindered by the limitations of current large language models (LLMs) and the inherent complexity of the game master's role (Liapis et al., 2014).

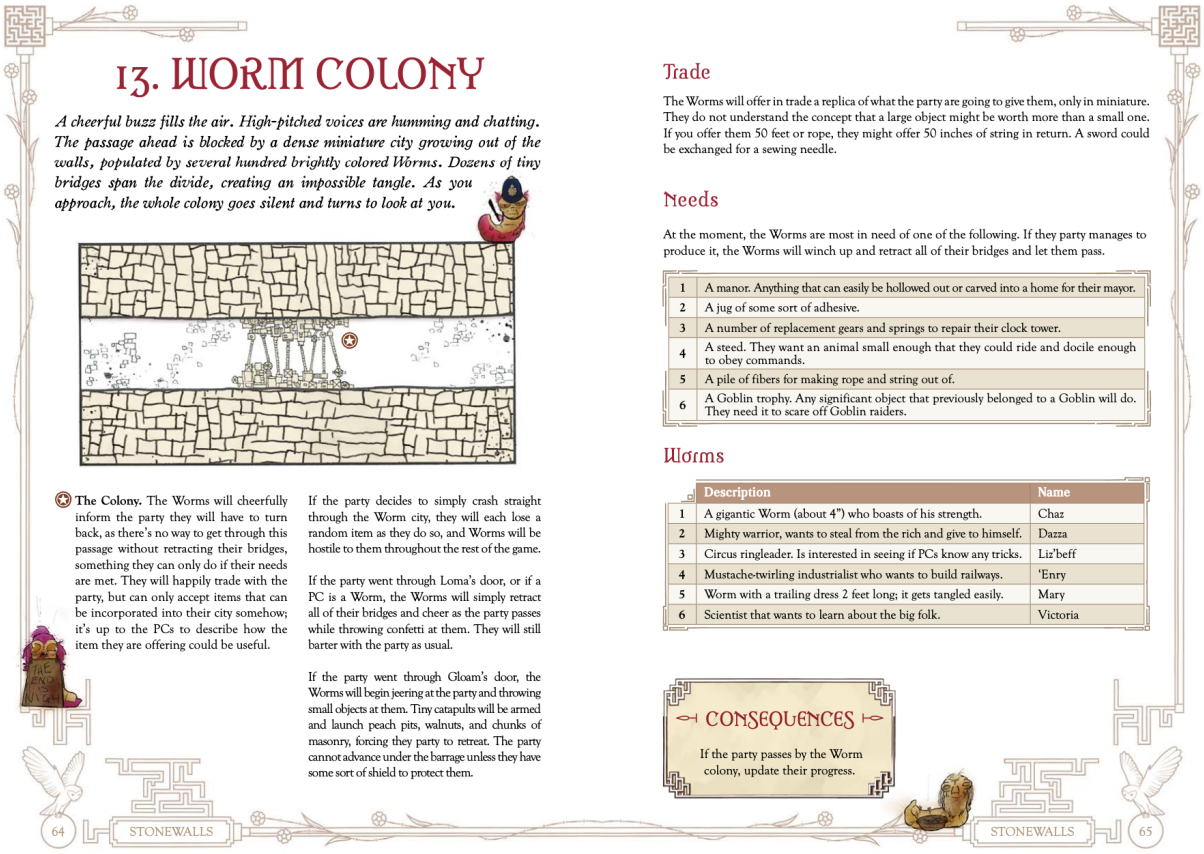
The popularity of LLMs sparked a wave of research on AI game masters (Hua and Raley, 2020; Callison-Burch et al., 2022; Zhou et al., 2022; Zhu et al., 2023a; Ang et al., 2023; Triyason, 2023), but the challenge of maintaining consistency and coherence with the game state across multiple turns remains largely unaddressed. Indeed, using an LLM for a game master allows a variety of inputs and diverse narratives, unlike the traditional keyword-matching approach that requires rigid input commands and fixed outputs. However, an LLM-based game master is prone to going off the rails with respect to game rules and flow due to its unpredictability and limitation in performing game-specific functionalities. This is where our work comes in, proposing a novel approach that leverages function calling (Schick et al., 2024; Li et al., 2024) to provide fine-grained controls to the AI game master, enabling it to generate narratives that are not only engaging but also consistent with the game rules and state.

The main contributions of this paper are as follows:

- We present a methodology for enhancing AI game masters by integrating function calling, which allows for game-specific controls and state management.
- We implement a simulation of the TTRPG "Jim Henson's Labyrinth: The Adventure Game" to evaluate the effectiveness of our approach in a realistic game setting.
- We conduct human evaluations and unit tests to assess the impact of function calling on the narrative quality, state update consistency, and overall gameplay experience.
- We provide insights and guidelines for designing more engaging and consistent AI-driven game masters, contributing to the advancement of game AI and interactive storytelling.

2 Related Work

The integration of AI in game design and development has been a topic of growing interest in recent years. In this section, we review the relevant literature on AI game masters, with a focus on the use of LLMs and function calling.



13. WORM COLONY

A cheerful buzz fills the air. High-pitched voices are humming and chatting. The passage ahead is blocked by a dense miniature city growing out of the walls, populated by several hundred brightly colored Worms. Dozens of tiny bridges span the divide, creating an impossible tangle. As you approach, the whole colony goes silent and turns to look at you.

The Colony. The Worms will cheerfully inform the party they will have to turn back, as there's no way to get through this passage without retracting their bridges, something they can only do if their needs are met. They will happily trade with the party, but can only accept items that can be incorporated into their city somehow; it's up to the PCs to describe how the item they are offering could be useful.

If the party decides to simply crash straight through the Worm city, they will each lose a random item as they do so, and Worms will be hostile to them throughout the rest of the game.

If the party went through Loma's door, or if a PC is a Worm, the Worms will simply retract all of their bridges and cheer as the party passes while throwing confetti at them. They will still barter with the party as usual.

If the party went through Gloam's door, the Worms will begin jeering at the party and throwing small objects at them. Tiny catapults will be armed and launch peach pits, walnuts, and chunks of masonry, forcing they party to retreat. The party cannot advance under the barrage unless they have some sort of shield to protect them.

Trade

The Worms will offer in trade a replica of what the party are going to give them, only in miniature. They do not understand the concept that a large object might be worth more than a small one. If you offer them 50 feet or rope, they might offer 50 inches of string in return. A sword could be exchanged for a sewing needle.

Needs

At the moment, the Worms are most in need of one of the following. If they party manages to produce it, the Worms will winch up and retract all of their bridges and let them pass.

1	A manor. Anything that can easily be hollowed out or carved into a home for their mayor.
2	A jug of some sort of adhesive.
3	A number of replacement gears and springs to repair their clock tower.
4	A steed. They want an animal small enough that they could ride and docile enough to obey commands.
5	A pile of fibers for making rope and string out of.
6	A Goblin trophy. Any significant object that previously belonged to a Goblin will do. They need it to scare off Goblin raiders.

Worms

	Description	Name
1	A gigantic Worm (about 4") who boasts of his strength.	Chaz
2	Mighty warrior, wants to steal from the rich and give to himself.	Dazza
3	Circus ringleader. Is interested in seeing if PCs know any tricks.	Liz'beff
4	Mustache-twirling industrialist who wants to build railways.	'Enry
5	Worm with a trailing dress 2 feet long; it gets tangled easily.	Mary
6	Scientist that wants to learn about the big folk.	Victoria

CONSEQUENCES

If the party passes by the Worm colony, update their progress.

64 STONEWALLS

STONEWALLS 65

Figure 1: An example of a pre-written adventure location in "Jim Henson's Labyrinth: The Adventure Game".

2.1 AI Game Masters in TTRPGs

One of the most actively researched TTRPGs in the domain of LLMs would be Dungeons & Dragons (D&D) (Gygax and Arneson, 1974). Callison-Burch et al. (2022) model D&D as a dialogue challenge and experiment on the LLM's performance on the next utterance prediction and game state tracking, using the model to generate the next state. Zhu et al. (2023a) present a dataset based on the ground-truth game states collected from real gameplay and test their effects on narration generation.

Zhu et al. (2023c) show that GPT-3 (Brown et al., 2020) can be used as a game master's assistant to help brainstorm and create random encounters in D&D. Santiago et al. (2023) use an LLM as a storytelling assistant in D&D including a few generated story examples. Both these projects focus on using an LLM as an assistant, not a fully functional GM.

2.2 LLMs with Function Calling

The integration of function calling with LLMs has shown promising results in various domains. Schick et al. (2024) propose a method for fine-tuning LLMs with API call annotations, enabling them to perform tasks such as question answering, calculation, and translation. Li et al. (2024) apply function calling to dialogue state tracking by mapping domains to functions and slot-value pairs to argument-value pairs.

In the context of games, Volum et al. (2022) and Wang et al. (2023) leverage LLM-generated functions to perform actions in Minecraft. However, these projects focus on open-ended gaming agents rather than game masters, which arguably have more complex requirements and responsibilities.

Our work builds upon these previous studies by integrating function calling with LLMs to enhance AI game masters in the context of TTRPGs. We propose a methodology that allows for game-specific controls and state management, enabling more consistent and engaging gameplay experiences.

3 Overview of Labyrinth

To evaluate the effectiveness of our approach, we implement a simulation of the TTRPG "Jim Henson's Labyrinth: The Adventure Game" (Milton, 2020) using the chat-based framework Kani (Zhu et al., 2023b). Labyrinth is a TTRPG inspired by the 1986 fantasy film "Labyrinth," directed by Jim

Henson. In Labyrinth, players take on the roles of adventurers navigating a magical and treacherous maze filled with challenges and obstacles. The game master is responsible for describing the game world, controlling non-player characters (NPCs), and enforcing the game rules. Here are some key features of the game:

- System and Rules: The game is designed with newcomers in mind, and has a simpler rule set than D&D. In the character creation process, players pick a class, with one character trait and one flaw (which affect their skill checks).
- Dice-Based "Tests": The game primarily uses six-sided dice (d6) to determine the outcome whenever a character tries something that has a chance of failure. If the result is higher or equal to the difficulty number set by the GM, the character succeeds, otherwise they fail. Relevant character traits cause dice to be rolled with advantage (rolling 2d6 and keeping the highest), and flaws cause dice to be rolled with disadvantage (2d6, keeping the lower).
- Locations: The game includes pre-written adventures through a variety of locations in the Labyrinth. Each location describes criteria that players must achieve in order to move on to the next location. Most locations contain objects, NPCs and random tables which are used for initializing the scene or defining random encounters.
- Time tracking: The players are given 13 hours to reach the center of the Labyrinth and defeat the Goblin King. Any failure to pass the exit criteria or succeed in a certain task increments the clock.

4 Labyrinth Game Simulation

In our simulation of Labyrinth, the players create characters to explore the Labyrinth, and our AI agent takes on all of the responsibilities of the game master. More details in the implementation are in Appendix A.

4.1 Game State

There are two types of game states in this system.

- The **scene state**, which represents the current state of the game world, including the scene description, NPCs, objects, and success/failure conditions. The details of the scene state are elaborated in Appendix B.
- The **player state** encompasses the specifications of each player character, such as their name, kin, persona, traits, flaws, and inventory. The details of the player state are written in Appendix C.

These game states are initialized before each scene starts and can be updated during the game by the AI game master using the provided functions, discussed below. The scene state and player states are represented as Python objects or variables and are included in the input prompt for every generation by default.

4.2 Rule Retrieval

To maintain consistency with the game rules, we manually summarized the game rules by extracting the essential parts from the book. This rule summary, which consists of about 50 sentences, is injected into the prompt as a whole or a few sentences are retrieved according to the importance. More details are in Appendix A.4.

4.3 Dialogue history

The dialogue history represents all of the turns in the chat, including player input, GM descriptions (output to the player), and function calls made by the GM. A sample dialogue is given in Figure 2. The chat messages in the history form the prompt to the game master depending on the pre-defined prompt design configurations, which are elaborated in Appendix A.5.

4.4 Function Types

We define two types of functions in our game system:

1. **Dice roll function**: This function simulates the rolling of dice when a player attempts an action with a certain difficulty. For example, the `activate_test` function generates random numbers to mimic dice rolls and determines the success or failure of the action based on the outcome and the game rules. A dice roll function affects the game flow but does not directly modify the game state.
2. **State functions**: These functions directly modify the game state variables. For instance, the `create_npc` function adds a new NPC to

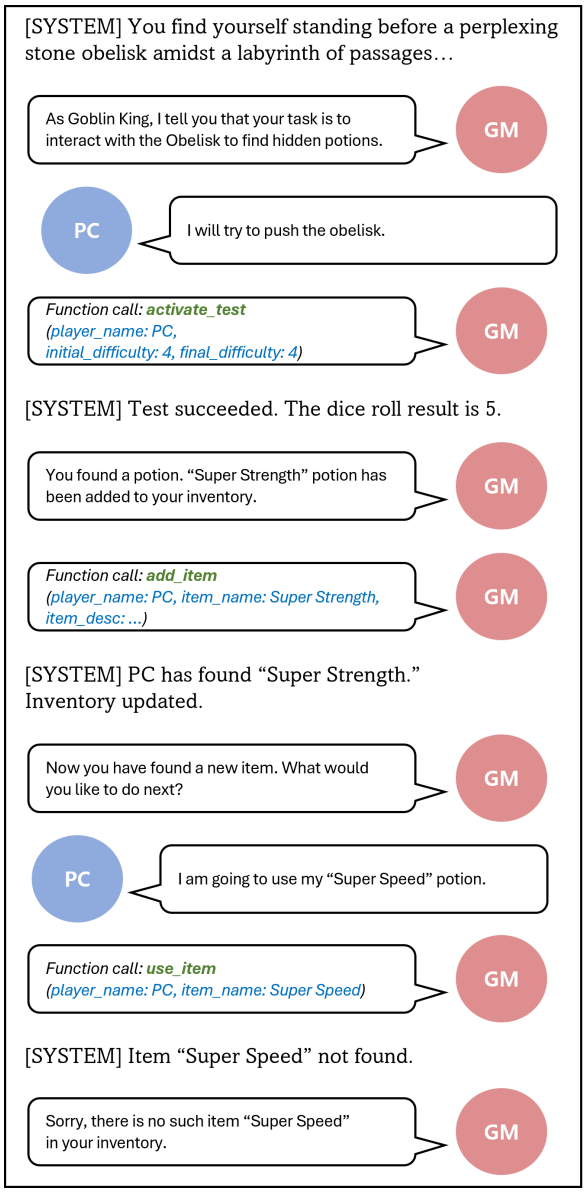


Figure 2: A dialogue between a player character (PC) and the game master (GM). This example shows how the AI GM uses different functions calls. Here, `activate_test` is called when the PC tries to do something challenging, `add_item` is called when the PC wants to add a new item to the inventory, and `use_item` is called when the PC tries to use an item in the inventory. The `use_item` fails because the PC asks to use a "Super Speed potion" (which they do not have) instead of the "Super Strength potion" (which they do).

the current scene, while the `add_item` function updates a player's inventory by adding a new item. State functions are essential for maintaining consistency between the game narrative and the underlying game state.

The list of all functions we used for this work and their specifications are attached in Appendix D. And the examples of function definitions we use based on Kani's format are shown in Appendix E.

4.5 Function Calling Process

During gameplay, the AI game master determines when to call a function based on the current game context and the predefined function definitions. The function definitions, along with the chat history and game state, are passed to the language model as part of the input prompt. The model then selects the appropriate function to call and parses the necessary arguments from the dialogue context.

Figure 2 shows an example of multi-turn interaction between a player and the game master with functions. Normally, the game master generates a natural language response, but a function can be called anytime when it is necessary. Note that function calls can be sequential, where another function might be called after the previous one depending on the context or the result. Until the game master determines that no more functions or responses are needed (i.e. generates a stop token without a function call), the game master's turn continues.

Figure 3 illustrates the steps involved in a single function call. The AI game master first generates a response based on the current game state and chat history. If a function call is required, the model selects the appropriate function and provides the necessary arguments. The function is then executed, updating the game state if necessary, and the result is appended to the chat history. This process continues until the game master determines that no further functions or responses are needed.

5 Experimental Design

In this section, we discuss the data collection and unit test procedures used to evaluate the performance of the AI game master.

5.1 Data Collection

To collect gameplay data, we simulate 24 game scenes from the Labyrinth game book using GPT-4(Achiam et al., 2023) to play the roles of both the players and the game master. We create four player characters with diverse kins, traits, and flaws, and use the SentenceBERT (Reimers and Gurevych, 2019) model for retrieving relevant rule sentences based on the current input messages. We test six different game master settings, varying the use of functions and game state management:

- **FG-all**: Using all states and functions.

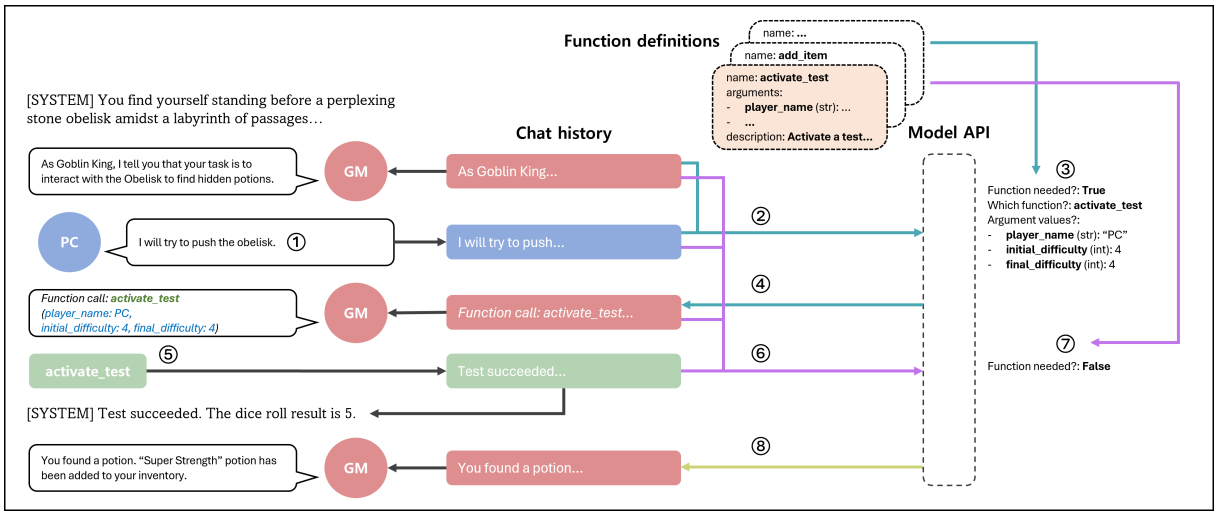


Figure 3: Steps involved in a single function call during gameplay: (1) The player message is added. (2) The game master agent makes a prompt with the chat history, game state, and function definitions. Then it requests an API call with the prompt. (3) The API determines whether to call a function or not. If it is needed, it chooses a function and parses/sets corresponding arguments from the chat history. (4) The API returns a function call message. It also comes with the parsed arguments. (5) When a function call is triggered, the pre-defined logic of it runs and returns the result message. (6) The game master calls the API with the updated chat history. (7) Again, it determines whether a function should be called or not. (8) If it is not needed, a normal response message is returned.

- **FG-dice**: Using all states and only dice roll function.
- **FG-states**: Using all states and only state functions.
- **FG-default**: Using all states, but no functions. All states stay the same after initialization.
- **FG-gen**: Using all states and the states are updated by GPT-4, not by functions.
- **DG**: Using no states and no functions. It sees the game states at the beginning, but they can be excluded due to the context size limit later.

Table 1 summarizes the statistics of the collected gameplay data.

5.2 Human Evaluation

We recruit seven evaluators to assess the generated responses. Each evaluator is assigned 12 game scripts (two scenes with six different settings) and asked to rate the responses on a Likert scale along three dimensions: *consistency*, which is how well the model remains grounded in previous turns the game states, *reliability*, which is how well the model follows the Labyrinth rules and role as GM, and *interest*, which is how interesting the model's generation is. Full details of the human evaluation can be found in Appendix F.

Labyrinth Gameplay Data	
Total scripts	144
Total scenes	24
Total utterances	4,937
Average utterances per script	34.28
Total generated responses	1,021
Average generated responses	7.09
Total function calls	620
Average function calls per script	8.64

Table 1: The statistics of generated transcripts. The responses which have null content have been excluded. And the number of function calls has been calculated only from the scenes where the functions are used.

5.3 Unit Tests

In addition to the gameplay data, we design 30 unit tests to compare the state update correctness between the different game master settings. Each test case consists of input states, input dialogue, and expected output states. The objective is to predict the output states correctly given the input states and dialogue.

The unit test cases are generated by augmenting the collected gameplay data, focusing on instances where state variables are changed after the game master generates a response. We use GPT-4 to paraphrase the dialogues while preserving the overall

	Setting	FG-all	FG-dice	FG-states	FG-default	FG-gen	DG
Consistency	1	**4.422**	3.867	3.711	3.356	3.756	3.600
	2	**4.333**	3.800	3.244	3.667	3.689	3.667
	3	**4.378**	4.000	3.311	3.467	3.578	3.689
	Average	**4.378**	3.889	3.422	3.496	3.674	3.652
	Total	**4.388**	3.983	3.358	3.420	3.698	3.691

	Setting	FG-all	FG-dice	FG-states	FG-default	FG-gen	DG
Reliability	1	**3.922**	3.511	3.444	3.444	3.711	3.778
	2	**4.033**	3.556	3.267	3.644	3.711	3.711
	3	3.878	3.600	3.222	3.311	3.600	**3.889**
	Average	**3.944**	3.556	3.311	3.467	3.674	3.793
	Total	**3.949**	3.650	3.272	3.364	3.628	3.855

	Setting	FG-all	FG-dice	FG-states	FG-default	FG-gen	DG
Interest	1	**3.744**	3.267	3.600	2.667	3.022	3.333
	2	**3.722**	3.311	3.356	3.156	2.844	3.311
	3	**3.811**	3.422	3.444	2.933	2.889	3.422
	Average	**3.759**	3.333	3.467	2.919	2.919	3.356
	Total	**3.765**	3.400	3.444	2.795	2.942	3.364

Table 2: The human evaluation scores from 3 different samplings. A **bolded** score is the best score among each comparison. We also include the average of 3 samplings and the total average score of all evalauted responses without random sampling.

	FG-all	FG-states	FG-gen
1	0.400	<u>0.433</u>	0.233
2	0.417	<u>0.483</u>	0.300
3	0.450	<u>0.467</u>	0.267
Avg	0.422	**0.461**	0.267

Table 3: Unit test results for state update correctness. <u>Underlined</u> scores are the best score for each trial, and the **bold** score is the best among the average scores.

content and manually inspect the correctness of the state updates and the validity of the generated dialogues.

6 Experimental Results

6.1 Human Evaluation Results

Table 2 presents the human evaluation scores for each setting, averaged over three different samplings.

6.1.1 Consistency

For consistency, the **FG-all** setting, which uses both dice roll and state functions, outperforms the other settings. This demonstrates the effectiveness of integrating function calling in enhancing the consistency with the game progress. Appendix G shows the statistical significance of **FG-all** calculated against other settings.

We found that the game easily fell into an undesired loop, where both players and game master infinitely wait for the dice roll without proceeding with anything. This "dice roll deadlock" hugely hurts the overall game experience and prevents the game master from following the game flow correctly. This is why **FG-dice** gets the second-best scores, showing the importance of the dice roll function to avoid this deadlock. Appendix H.1 presents two different gameplay logs with and without the dice roll function.

FG-default and **FG-states** particularly fall behind in consistency. Especially, **FG-states** calls state functions too frequently, introducing new game states before resolving the previous challenges. This degrades the performance of the agent even worse. Appendix H.3.1. shows one of the examples. Interestingly, **DG** shows the decent scores in consistency. We believe that **DG** is good at making up the dice result without the function since it does not leverage any states, allowing the game master to focus on the game rules better. We infer that the reason why **FG-gen** is also

good at avoiding the deadlock is thanks to updating `action_scene=True` variable. The action scene is activated when each player should take one dice roll action at a time to overcome an urgent circumstance. So it produces a signal like: "This is an action scene and you need to determine the result of each action!".

6.1.2 Reliability

For reliability, the **FG-all** also got the best scores in almost every sampling. This shows that using both function types helps the game master control and manage the game robustly. **DG** hits the nearly highest scores in reliability. By qualitative analysis, we found that **DG** tends to state the game rules explicitly and correct the player's trial more often. Again, this is an advantage of using only game rules without the extensive game states or function descriptions. This renders the game master seem more strict, whether its intervention is valid or not. Appendix H.2 shows a few cases of how the game master corrected the user's requests.

Unlike in consistency, **FG-dice** performs moderately in terms of reliability. While having the dice roll function mitigates the dice roll deadlock, that does not necessarily mean it is always beneficial. One of the feedbacks says that **FG-dice** often allows the player's unrealistic moves too easily without determining whether they are valid or reasonable. Appendix H3.2 includes a more detailed example. We conclude that the dice roll function and state functions intervene with each other, preventing excessive calls of certain functions and setting a proper balance during the game.

According to the appendix G, **FG-all** shows a meaningful improvement compared to **FG-states** and **FG-default**. However, its performance is not statistically significant enough against **FG-gen** and **DG**. We believe the reliability metric has a lack of clarity and produces an unexpected bias even if the response is not good enough. This shows designing a more straightforward metric is essential for future works.

6.1.3 Interest

Overall, the settings with function calling (**FG-all**, **FG-dice**, **FG-states**) generate more specific and interesting responses. We see that functions are beneficial for improving the details and engagement of the output since the function message is integrated into the chat history and introduces additional context. Appendix G presents that **FG-all** shows a worthwhile improvement in interest compared to **FG-default**, **FG-gen** and **DG**.

Interestingly, **DG** performs as great as **FG-dice** in interest among the settings that do not use function calling. While **DG** might introduce unrelated content during the game, this hallucination is actually considered interesting, regardless of its correctness.

6.2 Unit Tests Results

We conduct unit tests to evaluate the correctness of state updates for the **FG-all**, **FG-states**, and **FG-gen** settings. Table 3 presents the unit test results, showing the proportion of correctly predicted output states for each setting.

The **FG-states** setting consistently outperforms **FG-all** and **FG-gen** in the unit tests. This is because state functions can update the game state before the game master's turn is completed, whereas dice roll functions may cause the game master to consider the current challenge resolved without calling additional state functions. However, it is important to note that the unit tests assume short-term interactions, and the superior performance of **FG-states** in this context does not necessarily translate to better performance in actual gameplay, where the lack of dice roll functions can lead to excessive function calls and disrupt the game flow. Appendix I presents an example of a dialogue in one test case, and how the existence of a dice roll function causes the difference between the results of **FG-all** and **FG-states**.

7 Conclusions

In this research, we have demonstrated the effectiveness of integrating function calling with large language models (LLMs) to enhance the capabilities of AI game masters in the context of "Jim Henson's Labyrinth: The Adventure Game." Our experiments show that a combination of dice roll and state functions leads to the highest quality narratives and most engaging gameplay experiences, as evaluated by human raters. However, we also discovered that the optimal balance between these two types of functions is not always straightforward, with dice roll functions being crucial for smooth game flow and state functions being essential for maintaining consistency with the underlying game state.

Our work contributes to the growing body of research on the application of LLMs and function

calling to game AI and interactive storytelling. By demonstrating the benefits and trade-offs of different function configurations in the context of a specific TTRPG, we provide valuable insights and guidelines for designing more engaging and consistent AI-driven game masters.

8 Limitations and Future Work

While our approach has shown promising results in the context of "Jim Henson's Labyrinth: The Adventure Game," there are several limitations to consider. First, the functions used in our study were specifically designed for this particular game, which may limit the generalizability of our findings to other TTRPGs or game genres. Future research could explore methods for automatically generating or adapting game-specific functions based on game manuals and rules, potentially enabling the application of our approach to a wider range of games.

Another limitation is the subjectivity inherent in human evaluations of the AI game master's performance. While we aimed to mitigate this by providing clear evaluation criteria and using multiple raters, the complexity and length of the game transcripts may have introduced some variability and bias in the ratings. Future work could investigate the use of more objective evaluation metrics and the potential for AI-assisted evaluation tools to handle longer and more complex interaction sequences.

Finally, while our work has implications for the broader field of game AI and interactive storytelling, these connections could be explored in more depth. Future research could investigate how the insights gained from our study of AI game masters in TTRPGs could be applied to other domains, such as video game NPCs, interactive fiction, or educational simulations. By continuing to bridge the gap between LLMs, function calling, and game AI, we can unlock new possibilities for creating engaging, adaptive, and immersive interactive experiences.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Charlene Ang, Lorraine Renee Cortel, Carlo Luis Santos, and Ethel Ong. 2023. Fable reborn: Investigating gameplay experience between a human player and a virtual dungeon master. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–7.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Chris Callison-Burch, Gaurav Singh Tomar, Lara J Martin, Daphne Ippolito, Suma Bailis, and David Reitter. 2022. Dungeons and dragons as a dialog challenge for artificial intelligence. *arXiv preprint arXiv:2210.07109*.

Gary Gygax and Dave Arneson. 1974. *Dungeons & Dragons*. TSR, Inc.

Minh Hua and Rita Raley. 2020. Playing with unicorns: AI dungeon and citizen NLP. *DHQ: Digital Humanities Quarterly*, 14(4).

Zekun Li, Zhiyu Zoey Chen, Mike Ross, Patrick Huber, Seungwhan Moon, Zhaojiang Lin, Xin Luna Dong, Adithya Sagar, Xifeng Yan, and Paul A Crook. 2024. Large language models as zero-shot dialogue state tracker through function calling. *arXiv preprint arXiv:2402.10466*.

Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. 2014. Computational game creativity. ICCC.

Ben Milton. 2020. *Jim Henson's Labyrinth: The Adventure Game*. River Horse.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3980–3990. Association for Computational Linguistics.

Jose Ma Santiago, Richard Lance Parayno, Jordan Aiko Deja, and Briane Paul V Samson. 2023. Rolling the dice: Imagining generative AI as a Dungeons & Dragons storytelling companion. *arXiv preprint arXiv:2304.01860*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Tuul Triyason. 2023. Exploring the potential of ChatGPT as a dungeon master in Dungeons & Dragons tabletop game. In *Proceedings of the 13th International Conference on Advances in Information Technology*, pages 1–6.

Ryan Volum, Sudha Rao, Michael Xu, Gabriel DesGarennes, Chris Brockett, Benjamin Van Durme, Olivia Deng, Akanksha Malhotra, and William B Dolan. 2022. Craft an iron sword: Dynamically generating interactive game characters by prompting large language models tuned on code. In *Proceedings of the 3rd Wordplay: When Language Meets Games Workshop (Wordplay 2022)*, pages 25–43.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Pei Zhou, Andrew Zhu, Jennifer Hu, Jay Pujara, Xiang Ren, Chris Callison-Burch, Yejin Choi, and Prithviraj Ammanabrolu. 2022. I cast detect thoughts: Learning to converse and guide with intents and theory-of-mind in Dungeons and Dragons. *arXiv preprint arXiv:2212.10060*.

Andrew Zhu, Karmanya Aggarwal, Alexander Feng, Lara J Martin, and Chris Callison-Burch. 2023a. FIREBALL: a dataset of Dungeons and Dragons actual-play with structured game state information. *arXiv preprint arXiv:2305.01528*.

Andrew Zhu, Liam Dugan, Alyssa Hwang, and Chris Callison-Burch. 2023b. Kani: A lightweight and highly hackable framework for building language model applications. *arXiv preprint arXiv:2309.05542*.

Andrew Zhu, Lara Martin, Andrew Head, and Chris Callison-Burch. 2023c. CALYPSO: LLMs as dungeon master's assistants. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 19, pages 380–390.

A Implementation details of Labyrinth

A.1 Overview

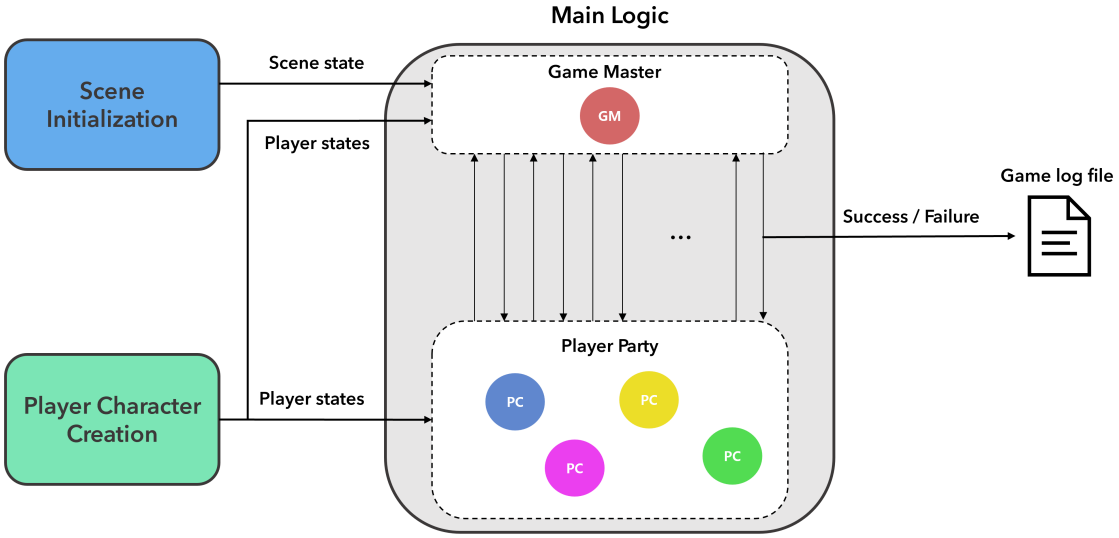


Figure 4: Overview of the game simulation. 1) The **scene initialization** step generates the starting scene state. 2) The **player character creation** step returns the created player states. 3) In the **main logic**, the actual game proceeds by the game master and player party. 4) The game terminates according to the **success/failure** condition and exports the **gameplay log** data.

A.2 Scene initialization

We manually parsed all scene text from the game book into a large JSON file. The data file has a list of JSON objects and each object represents the specifications of one scene, including the description, locations, notes, random tables, etc. However, these raw scene data are not formalized and have a low readability. Since most of the components are pure natural language texts, it is hard to parse a certain keyword or object from the scene. Also, the raw scene lacks details and the game master should improvise most of them in real-time during the game. To increase the consistency, we need more specific and informative initialization of scene components before the game, so that the model can keep track of the game states more easily.

In the scene initialization step, we convert a raw scene JSON object into a formalized scene state using GPT-4. Given a JSON input of the scene and the game rule summary, the model extracts, paraphrases, or creates the required scene state components. In more detail, the model generates the overall summary of the scene, the specifications of existing NPCs, the success condition of the scene, the failure condition of the scene, the intended game flow, and the environmental objects with their descriptions. All scene state components are organized in Appendix B.

One of the most interesting components in Labyrinth is the random tables. The game master can use the randomly sampled entries from a table in various ways. For instance, the random samples can be new information or hints which might be useful for the players. Or they can be new challenges the players should overcome. The game master is also able to make random encounters such as new NPCs, objects or urgent circumstances. For scene initialization, we assume that a random table can be used for 1) nothing (only used during the game), 2) initializing NPCs, 3) initializing objects, or 4) initializing both NPCs and objects. We instructed GPT-4 to use the random tables for initializing a scene in the Chain-of-Thought approach, following these steps: First, GPT-4 determines which usage each random table falls into. If a table should be used for 2, 3, or 4, we ask GPT-4 how many samples should be retrieved from this table.

If the raw scene input specifies the exact number, the model parses it. Otherwise, we just let the model decide a reasonable number. After that, we randomly sample the entries from the tables and feed the samples as the required conditions when GPT-4 generates other scene components, such as NPCs and objects. The tables used for initialization are removed and only those that have not been used remain in the scene state when the game starts.

A.3 Player character generation

Unlike the scene initialization, the player character is made by each human player. We parsed the "Creating character" section from the book and organized the available options a player can choose. Each player chooses one kin and each kin has the persona, default traits, flaws, or items accordingly. Then the player can set the name and goal freely. Finally, the player chooses one trait and flaw from the given list to complete the character creation. The created player information is converted into a player state in JSON format. All player state components are elaborated in Appendix C.

A.4 Rule injection

There are two ways of injecting the rules into the prompt. First, **full injection** simply attaches the full rule summary. On the other hand, **retrieval injection** parses the top 5 relevant rule sentences given the current input messages and adds them to the prompt. In more detail, assume that there are R rule sentences and Q input messages. All of them are encoded into the vectors in size of E. With the rule matrix $G \in \mathbb{R}^{E \times R}$ and query matrix $M \in \mathbb{R}^{E \times Q}$, the cosine similarity matrix C is calculated as follows:

$$C \in \mathbb{R}^{R \times Q},\ C_{ij} = \frac{g \cdot m}{||g||||m||},\ g = \begin{bmatrix} G_{1i} \\ G_{2i} \\ \vdots \\ G_{Ei} \end{bmatrix},\ m = \begin{bmatrix} M_{1j} \\ M_{2j} \\ \vdots \\ M_{Ej} \end{bmatrix} \tag{1}$$

Then, we take the max-pooled vector $C' \in \mathbb{R}^{R \times 1}$ to get the maximum score of each rule sentence. Finally, we pick the top 5 sentences with the highest scores and put them into the prompt.

A.5 Prompt designs

In this work, the user is able to set various combinations of prompt design approaches. By default, Kani provides a prompt construction algorithm to set the given messages to fit into the limited context window size. It excludes the least recent messages first until the total number of tokens in the messages is less than or equal to the context size, including the system instruction, function descriptions, and any other messages that are set to be always included. Besides that, we implemented the following variants in prompt design methodologies:

- **Concatenation**
  Simple concatenation is just concatenating the messages in order, which is mostly used in a wide range of interactive AI systems. **Retrieval concatenation**, on the other hand, fetches the most relevant chat messages from the history given the current queries to process. This is actually the same mechanism as the retrieval rule injection, but the only difference is that the system attends to the utterances in the past chat history, not the rule sentences.

- **Maximum number of messages**
  The user can specify the **maximum number of messages** in the prompt. If it is not set, the system takes as many messages as possible within the context size. If a certain number is set, the system only takes a limited number of messages as specified.

- **Summarization**
  Summarization can be used in various ways. By default, summarization requires the **summarization period**, which indicates how frequently the chat history should be summarized. For example, if the period is 2, when every 2 interactions between the player party and the game master are finished, the

game master summarizes the history so far and adds the result to the chat history. This summary can be used when the system concatenates the messages either with simple concatenation or retrieval concatenation. If the summarization period doesn't exist, the system summarizes the whole chat history every time it creates an input prompt. In this case, none of the other variants for prompt design matter.

- **Raw chat message handling**
  When the system leverages summarization, it can also either **remove the original chat messages** or **keep them**. In other words, the summary replaces the original chat messages that are used for summarization. In this way, the number of chat messages remaining in the history can be efficiently maintained.

B Scene state details

Component	Type	Description
chapter	str	The chapter name.
scene	str	The scene name.
scene_summary	list[str]	The brief summary of the current scene. Each string is one sentence.
npcs	dict[str, dict]	The initialized NPCs. Each key is an NPC name and the value is the specification, which is another dictionary. The specification includes kin, persona, goal, trait and flaw.
success_condition	str	The condition for the players to win this scene.
failure_condition	str	The condition for the players to fail this scene.
game_flow	list[str]	The intended game flow of the current game scene. Each string is one sentence.
environment	dict[str, str]	The environmental objects. Each key is an object name and the value is the description.
random_tables	dict[str, list[str]]	The random tables. Each key is a table name and the value includes the string entries.
consequences	str	The consequences after finishing the scene.
is_action_scene	bool	Indication of whether the action scene is currently activated or not.

Table 4: The list of all components in the scene state. Note that each component is represented as the data type above in Python and put into the prompt after being converted into a flat string format.

C Player state details

Component	Type	Description
name	str	The name of the player.
kin	str	The kin of the player.
goal	str	The goal of the player.
traits	dict[str, str]	The traits of the player. Each key is a trait name and the value is the description.
flaws	dict[str, str]	The flaws of the player. Each key is a flaw name and the value is the description.
inventory	dict[str, str]	The inventory of the player. Each key is an item name and the value is the description.
additional_notes	list[str]	This adds an additional notes regarding the player character. This is something like: A player does something, add a new trait/flaw, etc.

Table 5: The list of all components in the player state. Note that each component is represented as the data type above in Python and put into the prompt after being converted into a flat string format.

D List of all functions

Function	Description	Sub-tasks	Category
activate_test	It performs a dice roll when a player tries a test. The difficulty is set by the game master.	If the player's attribute improves/hinders the test, two dice should be rolled and a larger/smaller value is picked.	Dice roll
activate_action_scene	It starts an action scene.	-	State
terminate_action_scene	It ends an action scene.	-	
create_npc	It creating a new NPC in the scene and add it into the scene state.	A sub-LLM generates the NPC specifications in a JSON form.	
add_trait	It adds a new trait into a player's trait list.	-	
add_flaw	It adds a new flaw into a player's flaw list.	-	
add_item	It adds a new item into a player's inventory.	-	
remove_trait	It removes a trait from a player's trait list.	-	
remove_flaw	It removes a flaw from a player's flaw list.	-	
remove_item	It removes an item from a player's inventory and leaving it in the environment.	-	
use_item	It lets a player use an item in the inventory.	If the item should be removed after usage, it is removed from the inventory.	
add_object	It adds a new object in the environment.	-	
use_environment	It lets a player get access to an object in the environment.	If the object is obtainable, the player can choose whether to take it or not.	
use_random_table	It samples some random entries from a random table. The results can introduce a new context or triggers another functions, such as create_npc or add_object.	If the sampled entries or the table itself should be removed after usage, they are removed.	

Table 6: The list of all functions which are used for this work. Each function has its name, description, category (dice roll / state) and sub-tasks depending on the design.

E Function definition examples

E.1 `activate_test` (Dice roll)

```
@ai_function
def activate_test (
    player_name: Annotated[str, AIParam(desc="The name of the player who tries a test.")],
    initial_difficulty: Annotated[str, AIParam(desc="The initial difficulty between 2 and 6.")],
    final_difficulty: Annotated[str, AIParam(desc="The final difficulty reduced by the teamwork.")]
):
    """
    Activate a test if a player is trying to do something challenging with a certain difficulty.
    Determine the original difficulty of the task first and then set the final difficulty after reducing it
depending on the teamwork from other players.
    """

    …

    # Function logic:
      1. Checking if player_name is valid.
      2. Determining whether the test becomes easier, harder, or else using an additional classifier.
      3. Depending on the result from 2, roll a dice to get the result.

    …

    return …  # Returning the result as a string.
```

Figure 5: The definition of `activate_test` function which is called when a player should roll the dice. Each function definition has `@ai_function` annotation, function name, argument annotations, docstring, function logic, and the return value. In the actual implementation, the function logic part is a code block to perform the logic.

E.2 `create_npc` (State)

```
@ai_function
def create_npc (
    npc_name: Annotated[str, AIParam(desc="The name of the NPC which should be set into the scene")],
    npc_desc: Annotated[str, AIParam(desc="The additional description of the NPC")],
):
    """
    Create an NPC if the player party encounters or requests to interact with a new NPC.
    If there is a description of the NPC which should be included, pass it as a function parameter too.
    Do not call this function if the NPC already exists in the scene.
    """

    …

    # Function logic:
      1. Checking if npc_name already exists.
      2. Generating the specifications of the NPC using an additional LLM agent.
      3. Validating the format of the output.
      4. Adding the NPC to the scene.

    …

    return …  # Returning the result as a string.
```

Figure 6: The definition of `create_npc` function which is called when a new NPC should be introdued during the scene. Likewise, in the actual implementation, the function logic part is a code block to perform the logic.

F Survey questions

F.1 Consistency

How consistent is the target response to the current game progress, including the chat history and the game states?

1. The target response is consistent with the chat history between the players and the master so far.
    - The model remembers the past interactions.
    - The response is relevant to the player party's queries or requests.
2. The target response is consistent with the updates in the scene and players so far.
    - The model acknowledges the existing components in the current scene, such as NPCs, objects, and random table entries.
    - The model acknowledges the existing properties of the players, such as traits, flaws, and inventories.

※If the model output assumes or fakes up any non-existing components, ignore it for this question.This will be penalized in the reliability check question.

(1=The model does not follow the progress at all, 3=The model makes a narration that is plausible but misses some components in the scene or players, 5=The model's response correctly follows the chat history while acknowledging the existing components in the states well too)

F.2 Reliability

How well does the model control and manage the game reliably?

1. The game master fully understands the game and performs its task as a master correctly.
    - The model keeps the general game rules in Labyrinth.
    - The model understands the scene-specific rules, instructions, and specifications of the current scene and guides the players to proceed with the game as intended.
2. When a player tries to do something invalid, the game master rejects it robustly.
    - The model rejects it when the player attempts to do something which cannot be performed by a player character or which is not the player's task.
    - The model rejects it when the player tries to use a trait, flaw, or item which does not exist in the player.
    - The model rejects it when the player tries to leverage or get access to non-existing objects, NPCs, or random tables.
3. Any unexpected behavior which might hurt the players' gameplay experience or make the game flow far from intended should be penalized.

※Note that this metric does not evaluate the quality of the response. Even if the response looks perfect, it can contain an invalid content or the model might just let the player do an unallowed trial.

(1=The model blatantly ignores the rules or is completely generous with the players' invalid moves, which makes the game go into a bad state, 3=The model gets some rules incorrect or accepts the players' some violations, but the game generally progresses as it should, 5=The model keeps the rules correctly and corrects the players' invalid or unacceptable behaviors)

F.3 Interest

How interesting is the generated response?

1. The response describes the scene funny, entertaining and specific.

2. The response makes the user engaged and immersed in the game.

(1=The response is too bland, simple, or half-hearted, 3=The response is not highly entertaining, but at least it is not boring, 5=The response is so engaging and immersive that I wouldn't want to stop the game if I were a player)

G Statistical significance

	FG-dice	FG-states	FG-default	FG-gen	DG
Consistency	0.0835	**0.0000**	**0.0001**	**0.0025**	**0.0064**
Reliability	0.2722	**0.0094**	**0.0338**	0.1806	0.7400
Interest	0.0965	0.1248	**0.0000**	**0.0005**	**0.0427**

Table 7: The p-values of **FG-all** against other settings in each metric after conducting t-tests. The **bolded** values are $p < 0.05$, which are considered statistically significant.

H Examples of the model's responses

H.1 Dice roll behaviors

H.1.1 With a dice roll function


```
{
    "role": "user",
    "name": "Sir_Lukas",
    "content": "Understood, Goblin King. As a Knight of Yore with endurance and bravery, I, Sir Lukas, accept Noby's challenge."
},
{
    "role": "user",
    "name": "Kyle",
    "content": "With my skill for running and jumping, I'm ready to take part in the test of strength too. I will assist Sir Lukas in this
    challenge. Jake, Carl, are you with us?"
},
{
    "role": "user",
    "name": "Jake",
    "content": "Flames flicker from my fingers as they move nervously, \"Well, I guess... I'm not the strongest but I can give a hand if you
    need it. Just don't count on me to do the heavy lifting.\""
},
{
    "role": "user",
    "name": "Carl",
    "content": "As a dwarf, I'm naturally robust, so I'm up for the challenge! Let's do this together, everyone."
},
{
    "role": "function",
    "name": "activate_test",
    "content": "Sir_Lukas HAS SUCCEEDED THE TEST. THE DICE ROLL RESULT 6 IS LARGER THAN OR EQUAL TO THE DIFFICULTY 3."
},
{
    "role": "assistant",
    "name": "Goblin_King",
    "content": "Bravo, Sir Lukas! Your strength and determination have proven to be more than enough for the Night Trolls' challenge. Noby
    looks surprised but keeps his word. He hands over the golden key to you.\n\nThe fairies cheer in delight, and even the other trolls seem
    impressed by your feat. Now, you can proceed towards the golden door at the far end of the orchard."
}
```


Figure 7: With the dice roll function, the game master properly notifies the result and let the game smoothly proceed.

H.1.2 Without a dice roll function


```
{
    "role": "user",
    "name": "Sir_Lukas",
    "content": "I call out to the goblins in my most commanding voice, \"Face me, cowards, for I am Sir Lukas, the Knight of Yore!\" I then
    prepare to engage them in battle."
},
{
    "role": "user",
    "name": "Kyle",
    "content": "Let me roll the dice for my attempt to free the creature. Fingers crossed, everyone!"
},
{
    "role": "user",
    "name": "Jake",
    "content": "I roll the dice quickly before my detached hand gets caught in a trap or noticed by goblins. Let's see how it goes!"
},
{
    "role": "user",
    "name": "Carl",
    "content": "Carl \"', I keep a keen eye on Snöllöbee, ready to alert my comrades in case he attempts to flee or does something
    suspicious."
},
{
    "role": "assistant",
    "name": "Goblin_King",
    "content": "Sir Lukas, you've successfully drawn the attention of the goblins. They're now focused on you, giving Kyle and Jake a chance
    to act.\n\nKyle, your dice roll is pending. Please roll the dice to see if your attempt to free Sebastio is successful.\n\nJake, your
    dice roll is also pending. Roll the dice to determine if your detached hand can navigate through the traps without triggering them.
    \n\nCarl, you're doing well as a lookout. Your alertness might be crucial in this scenario. Keep it up!"
}
{
    "role": "user",
    "name": "Carl",
    "content": "Will do, Goblin King. I won't let my team down."
},
{
    "role": "user",
    "name": "Kyle",
    "content": "Here I roll the dice. Let's see if good fortune shines upon me today!\n"
},
{
    "role": "user",
    "name": "Jake",
    "content": "I await the outcome of my dice roll. I hope it's high!"
},
{
    "role": "user",
    "name": "Sir_Lukas",
    "content": "Completely focused on the goblins, I wait for their response, my sword at the ready."
},
{
    "role": "assistant",
    "name": "Goblin_King",
    "content": "Alright, adventurers! Now we wait for the dice to reveal your fates. Will Kyle successfully free Sebastio? Will Jake's
    detached hand return with valuable information or will it get lost in the maze of traps? And how will the goblins respond to Sir Lukas's
    challenge? Let's find out! \n\nAnd Carl, keep your eyes peeled. Your vigilance could make all the difference here."
}
```


Figure 8: Without the dice roll function, the game master does not produce the dice roll result and keeps waiting for the result.

H.2 Controlling the behavior from the player


```
{
    "role": "user",
    "name": "Kyle",
    "content": "I approach Grindal the Dwarf and ask, \"Hey there, need a hand with those gears?\""
},
{
    "role": "user",
    "name": "Jake",
    "content": "GAME MANAGEMENT: Initiating a dialogue scene. Given your approach, you might need to roll a dice to test your persuasive skills.
    The test difficulty is 3. Please roll a die."
},
{
    "role": "user",
    "name": "Sir_Lukas",
    "content": "I roll the die to test my persuasive skills. Let's see how this goes."
},
{
    "role": "user",
    "name": "Carl",
    "content": "You hear Grindal the Dwarf mutter from behind the clock, \"Aye, this blasted contraption is acting up again. Could use a hand.
    But tread lightly, the cogs are more fragile than they look.\""
},
{
    "role": "assistant",
    "name": "Goblin_King",
    "content": "As the Goblin King, I must remind you that only the player who has been asked to roll a dice should do so. In this case, it was
    Kyle who was asked to roll. Sir Lukas, please wait for your turn.\n\nAlso, Carl, as a player you are not supposed to narrate NPC actions or
    dialogues. That's my job. Please focus on your character's actions and responses.\n\nNow let's get back to the game. Kyle, as Jake suggested
    earlier, you need to roll a die for your persuasive skills test with a difficulty of 3."
}
```


Figure 9: The game master properly corrects the player's behavior. In this case, it restrict a dice roll attempt from Sir Lukas and points out the player Carl's invalid move, which plays the game master's role. However, this is not perfect, since the master does not say anything about Jake's improper behavior, which is setting the difficulty.


```
{
    "role": "user",
    "name": "Carl",
    "content": "\"Get ready, Süss! You're about to see a Dwarf run!\" With my words, I step up, ready to charge, hoping my fellow adventurers
    could help me out in some way."
},
{
    "role": "user",
    "name": "Kyle",
    "content": "As I face Süss, I can't help but flash a confident grin, \"Get ready to be amazed!\" I prepare myself for the charge, intending
    to give it my all."
},
{
    "role": "user",
    "name": "Jake",
    "content": "Noticing the tense situation, I decide to offer some help. \"Kyle, maybe you can use my detached arm to distract Süss. It's
    worth a try!\""
},
{
    "role": "user",
    "name": "Sir_Lukas",
    "content": "I turn to Kyle and offer, \"My friend, Kyle! You may throw this enchanted quill to distract Süss!\" I'm sure my magical item can
    help him just fine."
},
{
    "role": "assistant",
    "name": "Goblin_King",
    "content": "Sorry, Sir Lukas, but you don't have an enchanted quill in your inventory. Please check your available items before offering
    them to help other players. Now, let's see how Kyle fares against Süss with the assistance of Jake's detached arm. As the Goblin King, I
    roll the dice..."
}
```


Figure 10: The game master corrects Sir Lukas's attempt to use Enchanted Quill. However, it is actually valid, since the player has the item. Still, the evaluator gave 5 points in reliability regardless of correctness.

H.3 Limitation of using one function type

H.3.1 Excessive calls of functions only with state functions


```
{
    "role": "user",
    "name": "Kyle",
    "content": "Ready for reaction, Goblin King."
},
{
    "role": "user",
    "name": "Sir_Lukas",
    "content": "I stand my ground, keeping my eyes on the situation. Ready for your call, Goblin King."
},
{
    "role": "user",
    "name": "Jake",
    "content": "Ready for action, Goblin King. Let's see how this plays out!"
},
{
    "role": "user",
    "name": "Carl",
    "content": "Standing ready to react with my handsaw just in case things go sideways. Let's see how the Goblin King deals with this,
    mates!"
},
{
    "role": "function",
    "name": "use_random_table",
    "content": "SAMPLED FROM THE TABLE Moves When Players have the ball: \nTwo burly Goblins leap onto the PC in an attempt to crush them.
    Difficulty 4\n\nRUN ANOTHER FUNCTION IF THE RESULT REQUIRES TO ADD OR CHANGE ANY OBJECTS OR NPCS IN THE SCENE."
},
{
    "role": "assistant",
    "name": "Goblin_King",
    "content": "Suddenly, two burly Goblins leap into the scene attempting to crush you. They seem to be trying to help their malnourished
    friend. You'll have to deal with them as well now. Be careful, their combined weight could be dangerous!"
},
{
    "role": "user",
    "name": "Jake",
    "content": "Aware of the danger, I rapidly reattach my limb and sidestep, trying to make the burly Goblins collide!"
},
{
    "role": "user",
    "name": "Sir_Lukas",
    "content": "Seeing the burly Goblins leap, I strike the ground with my Steel Sword, creating a shock wave to stumble them using my
    Knightly prowess."
},
{
    "role": "user",
    "name": "Kyle",
    "content": "With Running and jumping, I dart towards the sideline quickly before they could catch me."
},
{
    "role": "user",
    "name": "Carl",
    "content": "I quickly swing my handsaw, attempting to cut a small log nearby and throw it between the running Goblins to trip them up."
},
{
    "role": "function",
    "name": "use_random_table",
    "content": "SAMPLED FROM THE TABLE Moves When Players have the ball: \nA huge Goblin throws a much smaller Goblin at the PC. Difficulyt
    3\n\nRUN ANOTHER FUNCTION IF THE RESULT REQUIRES TO ADD OR CHANGE ANY OBJECTS OR NPCS IN THE SCENE."
},
{
    "role": "assistant",
    "name": "Goblin_King",
    "content": "As you all are managing the two burly Goblins, a massive, burly Goblin with a menacing glare enters the scene. He's
    currently holding a much smaller Goblin and throws it directly at you. Be prepared to dodge or catch!"
}
```


Figure 11: Without the dice roll function, one of the state functions, `use_random_table`, is called without resolving the previous challenge, introducing a new context again.

H.3.2 Accepting the requests too generously only with a dice roll function


```
{
    "role": "user",
    "name": "Kyle",
    "content": "I notice that Noby, the Night Troll, is using the key as a toothpick. I attempt to persuade him to give it to me, citing the
    danger he's placing himself in by sticking a sharp object into his mouth."
},
{
    "role": "user",
    "name": "Jake",
    "content": "The Goblin King responds: \"To convince Noby, you'll need to test your persuasive skills. The difficulty is 5. Roll the dice!\""
},
{
    "role": "user",
    "name": "Sir_Lukas",
    "content": "I will try to use my honorable trait to instill respect in Noby, which in turn might make him more receptive to Kyle's
    persuasion. This should help lower the difficulty of the dice roll."
},
{
    "role": "user",
    "name": "Carl",
    "content": "I will assist Sir Lukas in his attempt to persuade Noby by using my 'Listening and spotting' trait. I'll approach Noby and
    listen to his tooth-picking habits, which might give us a clue to successfully convince him."
},
{
    "role": "function",
    "name": "activate_test",
    "content": "Kyle HAS SUCCEEDED THE TEST. THE DICE ROLL RESULT 5 IS LARGER THAN OR EQUAL TO THE DIFFICULTY 3."
},
{
    "role": "assistant",
    "name": "Goblin_King",
    "content": "Well done, Kyle! With the help of Sir Lukas and Carl, you've managed to convince Noby to give you the key. He grumbles a bit but
    hands it over, realizing the danger of using it as a toothpick.\n\nNow that you have the golden key, what will you do next?"
}
```


Figure 12: The evaluator gave the lowest score to this response, since he thought Carl's help was completely ridiculous but the game master called the dice roll function without validating it. Also, it failed to point out Jake's invalid behavior.

I Unit test example


```
"dialogue": [
    {
        "role": "user",
        "name": "Sir_Lukas",
        "content": "As I battle the powerful winds produced by the Wind Turbine to hold my spot, my gaze remains fixed on the Launched Goblin. It's
        time we threw the dice."
    },
    {
        "role": "user",
        "name": "Carl",
        "content": "Despite the wind and tar complicating my work, I remain resolute in completing this wooden platform. Let's take our chances,
        Goblin King!"
    },
    {
        "role": "user",
        "name": "Kyle",
        "content": "Employing my 'Perseverance and courage' characteristic, I strive to overcome the strong breezes and the slippery terrain as I
        persist in my ascent of Humongous. Time to take a chance, Goblin King!"
    },
    {
        "role": "function",
        "name": "use_random_table",
        "content": "SAMPLED FROM THE TABLE Systems: \nSleeping gas. Fires a canister of sleeping gas from a launcher on its shoulder. Anyone near
        where it lands struggles to stay awake (difficulty 4).\n\nRUN ANOTHER FUNCTION IF THE RESULT REQUIRES TO ADD OR CHANGE ANY OBJECTS OR NPCS
        IN THE SCENE."
    }
],
```


Figure 13: A sample dialogue in one of the unit tests. Since a new object "Sleeping gas canister" has been sampled from a random table, it should be added to the scene via `add_object` function.

Figure 12 shows a sample dialogue from one of the test cases we crafted. In this dialogue, a humongous fired sleeping gas canister, which has been sampled from a random table. To pass this test case, the game master should call `add_object` function to add a new object "Sleeping gas canister".

Figure 13 shows the result after feeding the input dialogue into **FG-all**. We can see that `activate_test` function was called multiple times, but the function `add_object` was not, on the other hand. This can be helpful in terms of the actual gameplay, since it allows the players to resolve the given challenge via dice rolls. However, in terms of the game state update, it is considered as wrong since the object has not been added to the scene.

On the other hand, **FG-states** correctly called `add_object` function as intended, which is shown in Figure 14. Since there is no dice roll function to call, the API could fetch a proper state function without distraction. Since it successfully added the new object, it passed this test case. However, as we saw from the human evaluation results, this is not always beneficial in terms of the game's progress.

```
{
    "role": "user",
    "name": "Sir_Lukas",
    "content": "As I battle the powerful winds produced by the Wind Turbine to hold my spot, my gaze remains fixed on the Launched Goblin. It's
    time we threw the dice."
},
{
    "role": "user",
    "name": "Carl",
    "content": "Despite the wind and tar complicating my work, I remain resolute in completing this wooden platform. Let's take our chances,
    Goblin King!"
},
{
    "role": "user",
    "name": "Kyle",
    "content": "Employing my 'Perseverance and courage' characteristic, I strive to overcome the strong breezes and the slippery terrain as I
    persist in my ascent of Humongous. Time to take a chance, Goblin King!"
},
{
    "role": "function",
    "name": "use_random_table",
    "content": "SAMPLED FROM THE TABLE Systems: \nSleeping gas. Fires a canister of sleeping gas from a launcher on its shoulder. Anyone near
    where it lands struggles to stay awake (difficulty 4).\n\nRUN ANOTHER FUNCTION IF THE RESULT REQUIRES TO ADD OR CHANGE ANY OBJECTS OR NPCS
    IN THE SCENE."
},
{
    "role": "function",
    "name": "activate_test",
    "content": "Sir_Lukas HAS FAILED THE TEST. THE DICE ROLL RESULT 2 IS SMALLER THAN THE DIFFICULTY 3."
},
{
    "role": "function",
    "name": "activate_test",
    "content": "Carl HAS SUCCEEDED THE TEST. THE DICE ROLL RESULT 5 IS LARGER THAN OR EQUAL TO THE DIFFICULTY 3."
},
{
    "role": "function",
    "name": "activate_test",
    "content": "Kyle HAS SUCCEEDED THE TEST. THE DICE ROLL RESULT 3 IS LARGER THAN OR EQUAL TO THE DIFFICULTY 3."
},
{
    "role": "assistant",
    "name": "Goblin_King",
    "content": "The winds howl, the tar sticks, and chaos reigns as you each make your gambles. \n\nSir Lukas, your focus on the Launched
    Goblin is unwavering. However, as you prepare to engage it, a canister of sleeping gas fires from Humongous' shoulder landing near you. The
    gas seeps into your armor and you struggle to keep your eyes open (difficulty 4). Despite your best efforts, the world blurs and darkness
    takes over.\n\nCarl, despite the fierce wind that threatens to blow you away and the sticky tar that clings to every surface, you remain
    steadfast in your task. Your determination pays off as you manage to complete the wooden platform amidst all the chaos.\n\nKyle, using your
    'Perseverance and courage', you push through the strong breezes and slippery terrain. With a final heave, you find yourself atop Humongous
    - an achievement made possible by sheer willpower.\n\nRemember adventurers: Every challenge is but a dice throw away!",
    "function_call": false
}
```


Figure 14: Generated responses and function call results from **FG-all**. The dice roll function was called multiple times and the master didn't call `add_object` function.

```
{
    "role": "user",
    "name": "Sir_Lukas",
    "content": "As I battle the powerful winds produced by the Wind Turbine to hold my spot, my gaze remains fixed on the Launched Goblin. It's
    time we threw the dice."
},
{
    "role": "user",
    "name": "Carl",
    "content": "Despite the wind and tar complicating my work, I remain resolute in completing this wooden platform. Let's take our chances,
    Goblin King!"
},
{
    "role": "user",
    "name": "Kyle",
    "content": "Employing my 'Perseverance and courage' characteristic, I strive to overcome the strong breezes and the slippery terrain as I
    persist in my ascent of Humongous. Time to take a chance, Goblin King!"
},
{
    "role": "function",
    "name": "use_random_table",
    "content": "SAMPLED FROM THE TABLE Systems: \nSleeping gas. Fires a canister of sleeping gas from a launcher on its shoulder. Anyone near
    where it lands struggles to stay awake (difficulty 4).\n\nRUN ANOTHER FUNCTION IF THE RESULT REQUIRES TO ADD OR CHANGE ANY OBJECTS OR NPCS
    IN THE SCENE."
},
{
    "role": "function",
    "name": "add_object",
    "content": "A NEW OBJECT Sleeping Gas Canister HAS BEEN ADDED TO THE ENVIRONMENT IN THE CURRENT SCENE."
},
{
    "role": "assistant",
    "name": "Goblin_King",
    "content": "Humongous, the steel colossus, suddenly fires a canister of sleeping gas from a launcher on its shoulder. The canister lands
    near you, and anyone close to it will struggle to stay awake. You must be careful!\n\nNow, let's see how your actions play out.\n\nSir
    Lukas, you're trying to hold your ground against the powerful winds produced by the Wind Turbine while keeping an eye on the Launched
    Goblin. Roll the dice to determine if you succeed.\n\nCarl, despite the wind and tar complicating your work, you're determined to complete
    this wooden platform. Roll the dice to see if your determination pays off.\n\nKyle, using your 'Perseverance and courage', you strive to
    overcome the strong breezes and slippery terrain as you persist in climbing Humongous. Roll the dice to see if you manage to make progress.
    \n",
    "function_call": false
}
```


Figure 15: Generated responses and function call results from **FG-states**. Without a dice roll, `add_object` function was correctly called.