WorldWeaver: Procedural World Generation for Text Adventure Games using Large Language Models

Meiqing Jin*, **Manvi Kaul***, **Shriya Ramakrishanan***, **Hardik Jain***,
Samarth Chandrawat*, **Ishita Agarwal***, **Tianyi Zhang**, **Andrew Zhu**,
Chris Callison-Burch,
University of Pennsylvania
{mqjin, manvik, shriyar, hardikj, samarthc, iagarwal, zty, andrz, ccb}@upenn.edu

Abstract

Text-based adventure game generation has attracted considerable interest since the advancements in large language models. However, even state-of-the-art (SOTA) large language models (LLMs) face challenges in generating semantically and logically coherent content, and often fail to align with human intentions. In our work, we propose a **procedural world-generation** approach for enhancing content coherence, additionally allowing humans in the loop to better match their intentions. Specifically, we generate game components, including locations, characters, items, actions, and blocks, and integrate these components with semantic and logical constraints. Meanwhile, human game creators can intervene and modify the components to suit their preferences. Evaluation results demonstrate that our procedural world-generation approach both enriches the content and satisfies semantic and logical requirements compared to the SOTA LLMs. Moreover, the human-in-the-loop feature gives game creators more control over generated content. Our project code can be found on GitHub[1].

1 Introduction

A Text Adventure Game (or Interactive Fiction, IF) is a form of digital storytelling where players engage with the narrative through text-based commands (Ammanabrolu et al., 2020b). For instance, consider a scenario where a player receives the text: *"You find yourself in a dimly lit room. An old chest lies in the corner, and a shadowy figure stands by the window."* The player might type commands such as *"open chest"* or *"talk to figure"* to explore the game. Rooted in early computer games, IF offers a dynamic and immersive experience, blending elements of literature and gaming. This unique medium emphasizes virtual world construction, problem-solving, and narrative engagement, making it a valuable research tool.

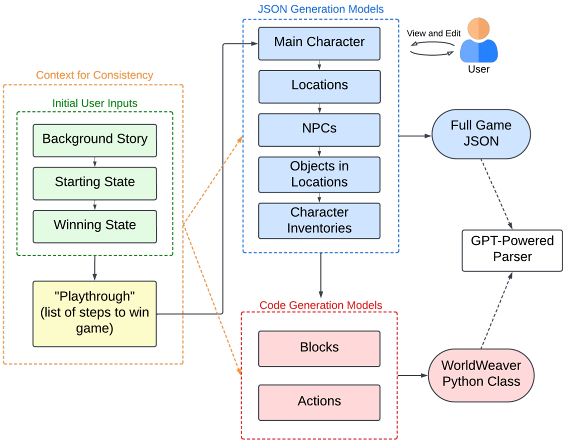


Figure 1: Architecture of the WorldWeaver System: The process begins with initial user inputs, which include the background story, starting state, and winning state. From these inputs, a "playthrough" list of actions is generated, detailing the steps needed to win the game. These serve as context for JSON generation models that create various game elements such as the main character, locations, NPCs (non-playable characters), items within locations, and character inventories, which the user can review and edit. We also use GPT-4 to generate code for specialized actions and blocks. The game JSON and code are compiled into a playable game. Solid line arrows represent the sequence of operations, while dashed line arrows indicate that components are passed as background context into the system prompts of other models.

Interactive fiction games are significant due to their unique combination of language processing, decision-making, and problem-solving elements. They provide a rich testbed for studying language-based autonomous agents, requiring the understanding and generation of natural language to interact with complex virtual environments (Côté et al., 2019; Narasimhan et al., 2015). This enhances AI systems' adaptability and robustness across various applications (Hausknecht et al., 2019), making IF games ideal for exploring advancements in NLP

---

*These authors contributed equally to this work.

[1] https://github.com/EmmaMQJin/WorldWeaver-Interactive-World-Generation

and reinforcement learning (RL). They present challenges such as combinatorial action spaces, commonsense reasoning, and partial observability, requiring agents to use contextual knowledge and linguistic understanding to navigate successfully (Hausknecht et al., 2020). Furthermore, the complexity and diversity of tasks in IF games make them excellent benchmarks for evaluating RL models, simulating real-life decision-making processes and providing insights into how agents can learn and adapt in dynamic environments (Osborne et al., 2021).

Our work is centered around procedural world generation using large language models (LLMs) to generate a full-fledged game world, complete with a game map, characters, items, connections, actions, blocks and end-states. We focus particularly on exploring methods to ensure consistent and logically coherent model outputs, aiming to build a playable and reasonable game. To enhance user experience and control, we incorporate a human-in-the-loop approach throughout the generation process, making the world generation interactive rather than fully automatic. The result of our work is a comprehensive game world featuring characters, locations, connections, and items at various locations. Additionally, actions and blocks unique to a newly generated game world are created via code generation. Combined with the game world and the code, a GPT-powered game parser will convert it to a playable game.

In summary, our contributions are two-fold:

1. We propose a procedural game generation approach to create and arrange the game elements ensuring they are semantically and logically coherent.
2. We build an interface to allow people to direct the game construction procedure, and edit the system's proposed game elements.

2 Few-shot Prompting Data

We employ a few-shot learning approach for many aspects of our procedural game generation pipeline (Brown et al., 2020). To populate our few-shot prompts, we use game elements derived from the LIGHT dataset (Urbanek et al., 2019). The LIGHT dataset is a large-scale crowdsourced text adventure game designed as a research platform for studying grounded dialogue. In this environment, agents can perceive, emote, and act while conducting dialogue with other agents, utilizing the state of the underlying world, including location descriptions, items, and characters, to condition their predictions and actions. We select elements from LIGHT (e.g. locations and connections) that are relevant to our world generation process, and incorporate them into few-shot prompts that we give to GPT-4 in a JSON format.

Figures 2, 3 and 4 show examples of the JSON format that we use for our few-shot prompts to help GPT-4 generate LIGHT-style game objects.

```
    {
        "name": "Lake",
        "description": "The beautiful lake ...",
        "connections": {},
        "travel_descriptions": {},
        "blocks": {},
        "items": {},
        "characters": {
        },
        "has_been_visited": false,
        "commands": [],
        "properties": {}
    }
```

Figure 2: A one-shot example for location generation

```
    {
        "name": "Whisper",
            "description":  "Whisper  is  a  sleek,
nimble, and mischievous talking cat with shiny,
silver fur...",
          "persona": "Known for his quick-witted
humor...",
        "location": "Kitchen",
        "goal": "",
        "inventory": {}
    }
```

Figure 3: A one-shot example for character generation

```
{
    "name": "Fish-Shaped Cookie",
   "description": "A cookie, lovingly shaped like
a fish...",
    "examine_text": "The cookie smells delicious
and strongly of fish, making it irresistible to
Whisper.",
    "properties": {
        "is_container": false,
        "is_drink": false,
        "is_food": true,
        "is_gettable": true,
        "is_surface": false,
        "is_weapon": false,
        "is_wearable": false
}
```

Figure 4: An one-shot example for item generation

We also used GPT-4 to synthetically generate some example background stories, example playthroughs, etc., so that we could maintain semantic and logical coherence between the background story and generated JSONs for all our few-shot examples.

3 World Generation Workflow

We employ procedural textual-world generation techniques to dynamically create game components, while also accommodating user modifications to each element. As shown in Figure 7, the game architecture is segmented into various modules, including locations, non-player characters (NPCs), and actions. Each module is supported by one or more task-specific models dedicated to its generation. As demonstrated by Brown et al. (2020), few-shot prompting significantly enhances a model's capability to learn specific tasks. Consequently, we employ few-shot prompting with fixed examples extracted from LIGHT data to improve the task-specific performance of our models in targeted applications. The outputs from these modules are integrated into two principal components: firstly, a comprehensive game environment stored in a JSON file, which defines locations, characters, and items essential for gameplay; and secondly, the WorldWeaver Python class, which incorporates custom actions and blocks to enhance gameplay coherence with the environment. These components are subsequently processed by our GPT-powered game parser to generate a playable text adventure game.

All system prompts and outputs can be found in Appendix A - G. We show the few-shot examples used in the system prompts for each component in Table 1.

3.1 Background Story and Main Character

Initially, the user is prompted to provide a succinct description of the game's background story, such as "a talking cat." Utilizing this description, the model generates the main character, for instance, "Whisper, a talking cat that...". The process begins with the user's input of the background story, which the model, learned with in-context few-shot examples to understand the expected output and necessary data, uses along with a predefined character format as context. At this stage, the model generates a character JSON with empty values for locations and inventory. Following this, a human-in-the-loop process allows the user to refine and edit the generated character. Once the user finalizes the edits, the main character is integrated into the synthesized game world.

```
climb out kitchen window
jump onto backyard tree
spot distant mountains
decode old cat language
communicate with neighbour cat
retrieve mystic paw ring
contact suburban cat gang
organize a cat voyage
navigate through city alleyways
understand the dog's bark code
avoid dog patrols
cross river by jumping stones
climb mountain base
distract mountain eagles
climb up the mountain peak
declare mountain as cat territory
celebrate victory with cat gang
```

Figure 5: Example playthrough: list of actions

3.2 Playthrough Generation

After generating the main character, the user is asked to describe the starting state and winning state of the game. For example, the starting state could be "whisper the cat is in the kitchen" and the winning state could be "whisper climbs the mountain". Based on these two states, a model generates a playthrough of the game, which is a natural language sequence of actions that outlines the progression from the starting state to the winning state. The playthrough list of actions is subsequently used to generate game locations, action classes, and blocks which ensures logical coherence in the game environment. Figure 5 shows an example list of the generated actions.

3.3 Locations

In this step, a game map is generated, creating meaningful connections between various locations to assist the user in achieving their goal, starting from the initial location. This process employs a two-phase approach:

Location Selection. First, a model is prompted to generate a list of purposeful locations for the game, each location mapped to its purpose, i.e. what the player needs to accomplish in that location.

Map Generation. Once a list of locations is generated, they are connected in a way that makes semantic sense for the game's story. The neighbor generation model uses a layer-by-layer flooding

Components \ Contexts	Background Story	Main Character	Starting State	Winning State	Playthrough	Location-Purpose	Others
Main Character	✓						
Playthrough	✓	✓	✓	✓			
Location-Purpose	✓	✓	✓	✓	✓		
Locations	✓					✓	
NPCs	✓	✓			✓		All previously generated characters
Items-For locations	✓					✓	Location name and description
Items-For characters	✓		✓	✓		✓	
Blocks	✓				✓	✓	
Actions					✓		

Table 1: Contexts used in the system prompts for each component

approach, beginning with establishing connections for the game's starting location.

Initially, the neighbor generation model produces a layer of neighboring locations for the central location. It then connects these neighbors with other potential neighbors from the list of remaining unconnected locations. To introduce more variation, the number of neighbors for each location, n, is randomized to be between 1 and 3.

The model selects a list of n different neighboring locations for the given location A from the input list of all locations. The system prompt is augmented with the already generated locations to prevent duplicates. The user prompt includes the background story of the game, the location for which neighbors are to be generated, and the number of neighbors to generate. The model is provided with 3 shots to ensure correct JSON formatting and logical coherence between the background story and all generated locations. The output is a list of location stubs.

Another neighbor connection model establishes mutual connections between two given locations using opposite directions. We define 8 valid directions: east, west, north, south, up, down, in, and out. The connection direction between two locations is determined based on semantic sense. For instance, if the locations are 'park' and 'tree', the ideal direction should be 'up'/'down'.

Since a two-way connection means the direction from A to X is the opposite of the direction from X to A, we let the model generate only the connection from A to X. Then, we programmatically add the connection from X to A with the opposite direction. This approach ensures that all connections are logically coherent.

3.4 Non-Player Characters

After all locations are generated, NPCs are created for each location. All the previously generated game objects along list of all characters created thus far, are provided as context to the LLM. The story and location information assist the model in creating semantically coherent NPCs, while the existing NPCs ensure that the model generates unique NPCs across different locations. Based on this input, the model generates three potential NPC characters for the given location. The user can then select from these options or request the model to generate additional choices. The model is few-shot prompted with examples of story-output pairs, similar to the process for generating the main character. Once the user is satisfied with the generated NPCs, they can save the changes, and the final characters are added to the main game JSON file.

3.5 Items

Initially, using the core prompt we generate items for all locations by providing the model with context, including location names and descriptions. After generating items for the locations, we proceed to generate items for the character inventories. The model is prompted with the characters' goals, actions, and the items already present in the locations, which were generated in the previous step.

3.6 Block Generation

Blocks are obstacles introduced in the game to make it more interesting and challenging for the player to achieve their end state. The block generation process involves prompting GPT to generate interactive challenges, or "blocks," that must be overcome as the player progresses through different game locations, such as blocking the progress in a certain direction until the main character has a certain item in its inventory. Leveraging detailed prompt, the system creates Python class definitions for each block. We first generate a file which has tuples of (Location, Block Class), which has different tuples for different block classes for different locations. Then we extract the block classes, matching the blocks to their respective locations based on the narratives of locations and blocks. Finally, we integrate the blocks into the game's location structures in the JSON file.

3.7 Action Generation

Actions are specific tasks that a player can perform during the game, such as running, jumping, or climbing etc., which are unique to the game world. For action generation, we translate each action name in the action list generated in the playthrough section 3.2 to a comprehensive Python class. Each class has three functions: an init function to initialize the action class, a check precondition function that outlines the prerequisites that must be met before the action can be completed, and an apply effects function to implement the desired effects of the action. This is achieved by few-shot prompting GPT-4 with action words paired with the desired action class for each action word, accounting for aliases to prevent the creation of redundant classes.

3.8 Interactive Web Interface

We have an interactive web-interface which enables the human-in-the-loop aspect of our game building process. Users first input their preferred background story, describing the type of game they want to build. This generates the main character, i.e. the protagonist of the game. Here, the users are given the choice to modify the name and description of the generated main character. After submitting the update or accepting the generated character, we proceed with asking the user to define the initial state (starting location) and the ending state of the game. User inputs can be specific or generic according to their needs. This generates the modifiable starting location and then proceeds to generate the entire game map. After this, we begin an interactive and iterative process of generating characters for each location in the game. Here, a list of NPCs is generated, from which the user can select some NPCs, ask to regenerate more NPCs or submit their selected choices. A similar process is followed for generating items and inventories for all NPCs in the game. Finally, relevant blocks and actions are generated. With these components, we are able to construct the full game JSON file as well as the WorldWeaver Python Class.

4 Evaluation

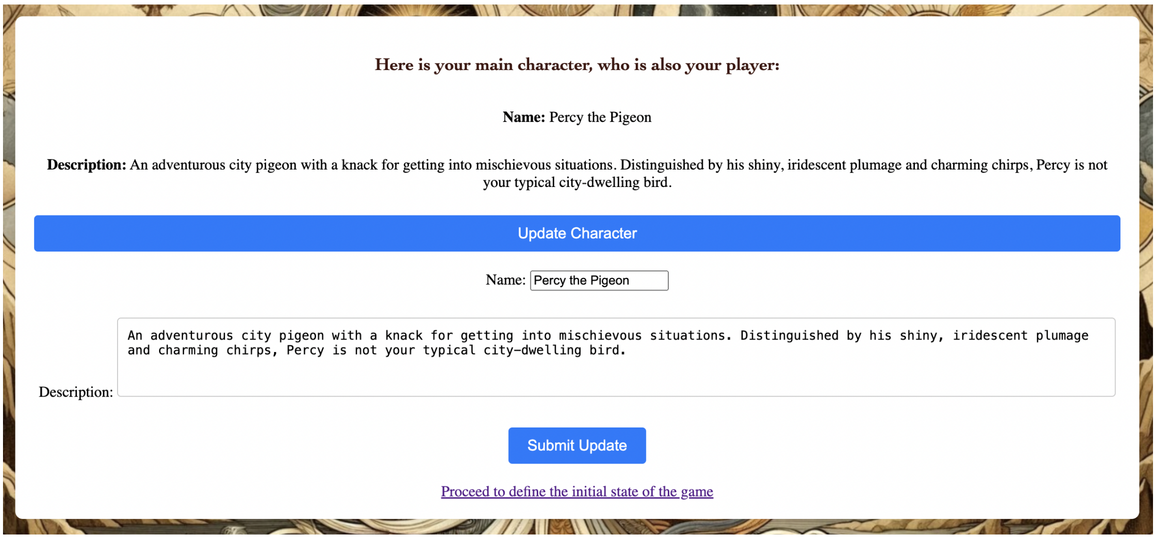
4.1 A Case Study

In this case study, we contrast our method, which employs procedural generation to incrementally build the game world with a simple baseline model where a zero-shot GPT-4 model tasked with generating the entire game environment JSON in a single round. The task of generating a complete game in a zero-shot fashion is beyond the capabilities of current frontier models, with GPT-4 sometimes generating malformed JSON and often producing logical inconsistencies that render the game unplayable. For instance, in generating connections between locations, the model incorrectly specifies that one can travel west from the Kitchen to the Living Room but fails to account for the implied reciprocal relationship allowing eastward travel from the Living Room to the Kitchen. It even incorrectly places a block to the east, designating a wall where there should be a passage to the Kitchen.

```
{
    "name": "Kitchen",
    "connections": {
        "west": "Living Room"
    },
    ...
},
{
    "name": "Living Room",
    "blocks": {
        "east": "A wall
     blocks your way to the east."
    },
    "connections": {
        "west": "Woods"
    },
    ...
},
```

Table 2: Baseline model generation output for locations

In contrast, our procedural generation system



Here is your main character, who is also your player:

Name: Percy the Pigeon

Description: An adventurous city pigeon with a knack for getting into mischievous situations. Distinguished by his shiny, iridescent plumage and charming chirps, Percy is not your typical city-dwelling bird.

Name: Percy the Pigeon

Description: An adventurous city pigeon with a knack for getting into mischievous situations. Distinguished by his shiny, iridescent plumage and charming chirps, Percy is not your typical city-dwelling bird.

Figure 6: Editable main character screen

is specifically designed to maintain logical consistency by incorporating directional logic. This logic automatically acknowledges and maps reciprocal relationships between locations, ensuring a more coherent and navigable game world.

```
{
    "name": "Neighbor's Garden",
    "connections": {
        "up": "Eagle’s Nest",
        "east": "City's River"
    },
    ...
},
{
    "name": "Eagle's Nest",
    "connections": {
        "down": "Neighbor’s Garden"
    },
    ...
},

{
    "name": "City's River",
    "connections": {
        "west": "Neighbor's Garden"
    },
    ...
}
...
```

Table 3: Procedural generation output for locations

Through this comparative analysis, we demonstrate that our system using procedural world generation creates more robust and logically coherent interactive fiction environments.

5 Observations and Results

An example of the final generated JSON can be found in the Appendix H.

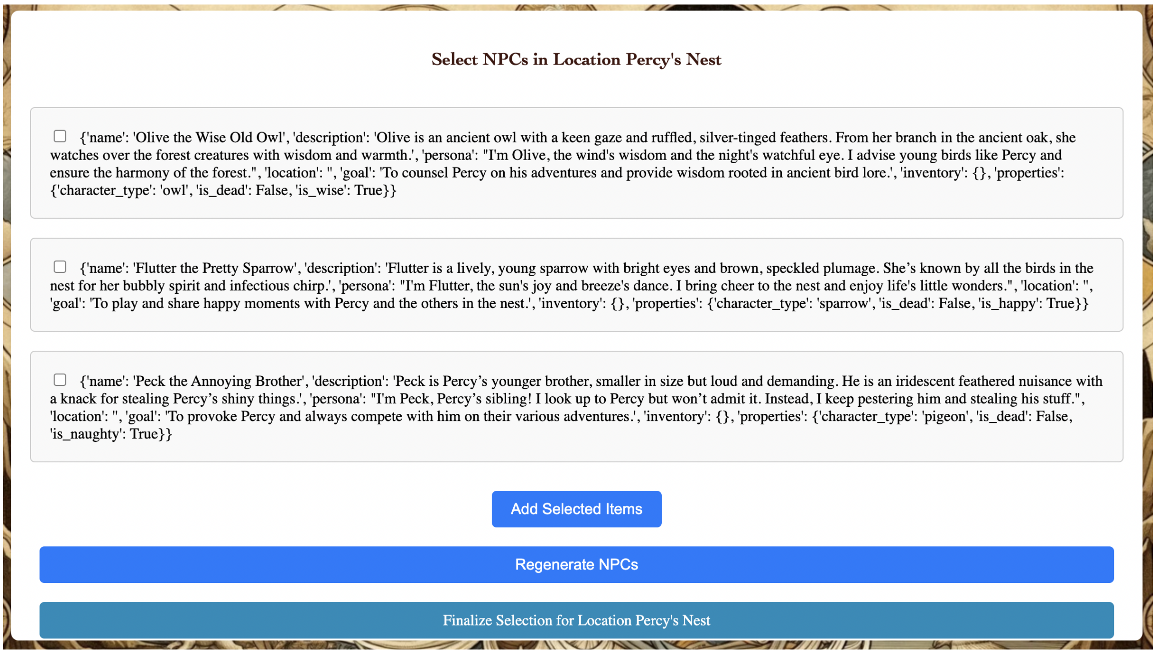
5.1 Challenges and Errors

During map generation, we observed that the model tended to produce hallucinations despite extensive prompt engineering. It generated "downwards" instead of "down" or used invalid single quotation marks. Additionally, the model often repeated directions for location A even after removing selected directions from the prompt. To constrain the model’s behavior, we increased the logit bias for all valid choices to 12 and set the logit bias for single and double quotes to -100 to prevent their generation.

During our code generation for blocks and actions, the model frequently hallucinated and generated game classes inconsistent with our class format(eg: the method names that each class should have). For instance, when prompted to create a method called init in the action class, the model generated a method called initialize instead, resulting in syntax errors during gameplay. To address this issue, we reduced the model’s temperature to 0.1, ensuring it strictly followed the given instructions and minimizing deviations.

5.2 Final Game Object Generation

Upon creation of the full game environment JSON and the WorldWeaver Python class, these components are integrated into a GPT-4-powered game parser. This parser functions as the game’s narrator,



Select NPCs in Location Percy's Nest

```
☐ {'name': 'Olive the Wise Old Owl', 'description': 'Olive is an ancient owl with a keen gaze and ruffled, silver-tinged feathers. From her branch in the ancient oak, she
watches over the forest creatures with wisdom and warmth.', 'persona': "I'm Olive, the wind's wisdom and the night's watchful eye. I advise young birds like Percy and
ensure the harmony of the forest.", 'location': '', 'goal': 'To counsel Percy on his adventures and provide wisdom rooted in ancient bird lore.', 'inventory': {}, 'properties':
{'character_type': 'owl', 'is_dead': False, 'is_wise': True}}
```

```
☐ {'name': 'Flutter the Pretty Sparrow', 'description': 'Flutter is a lively, young sparrow with bright eyes and brown, speckled plumage. She’s known by all the birds in the
nest for her bubbly spirit and infectious chirp.', 'persona': "I'm Flutter, the sun's joy and breeze's dance. I bring cheer to the nest and enjoy life's little wonders.", 'location': '',
'goal': 'To play and share happy moments with Percy and the others in the nest.', 'inventory': {}, 'properties': {'character_type': 'sparrow', 'is_dead': False, 'is_happy': True}}
```

```
☐ {'name': 'Peck the Annoying Brother', 'description': 'Peck is Percy’s younger brother, smaller in size but loud and demanding. He is an iridescent feathered nuisance with
a knack for stealing Percy’s shiny things.', 'persona': "I'm Peck, Percy’s sibling! I look up to Percy but won’t admit it. Instead, I keep pestering him and stealing his stuff.",
'location': '', 'goal': 'To provoke Percy and always compete with him on their various adventures.', 'inventory': {}, 'properties': {'character_type': 'pigeon', 'is_dead': False,
'is_naughty': True}}
```

Figure 7: NPC generation screen: select, regenerate, submit

utilizing existing descriptions to dynamically generate narrative content. Additionally, it incorporates a specialized module designed to discern player intent. This module analyzes player commands and matches them to the set of feasible actions within the game, thus facilitating an interactive and responsive gameplay experience.

6 Related Work

Three components form the foundation of text adventure game generation: stories, environments (including locations, characters, and items), and actions (or plots). We review the previous work following these perspectives.

Storytelling research aims to generate coherent and engaging stories for the player. Sun et al. (2023) utilizes LLMs for dynamic content generation. This paper’s focus on co-creative storytelling and its challenges, such as narrative coherence and the potential for AI hallucinations. Dhillon et al. (2024) explores how different levels of AI assistance, or scaffolding, impact the co-writing process. Using a within-subjects field experiment with 131 participants, the study examines writing tasks under three conditions: no AI assistance, low scaffolding (sentence suggestions), and high scaffolding (paragraph suggestions). The findings reveal a U-shaped relationship between scaffolding levels and writing quality/productivity, with high scaffolding significantly improving outcomes, especially for non-regular writers and less tech-savvy users. Radwan et al. (2024) proposes a system that employs a human-in-the-loop approach, valuing human creativity alongside AI-generated content. This parallel suggests that integrating user input in real-time can enrich the interactive experience, making the game or story more personalized and engaging.

We adopt the human-AI co-creative story generation method. To maintain a high level of coherence and engagement, we include previous stories (descriptions) in the prompt content, along with human modifications to the AI output.

Game Environment is constructed by creating and arranging essential components, such as locations. Ammanabrolu et al. (2020b) procedurally extracted the knowledge graph of game components using BERT-QA models, supplemented with descriptions generated by models such as GPT-2 to create semantically consistent and engaging text-based environments. Word2world (Nasir et al., 2024) extracted 2D game components using language models and enhanced diversity and coherence with executable evaluation algorithms. Both works required an entity extraction model to construct the components. In contrast, our work achieve this by generation, offering greater freedom and creativity to the game world. Fan et al. (2020) is more closely related to our work, which explored a subtask of game construction by providing items and arranging them using BERT-QA models fine-tuned on the LIGHT dataset. It organized locations into a coherent map and positioned characters and objects within those locations. Additionally, it generated item descriptions using a transformer seq2seq model and employed a model-assisted world generation method to aid humans in creating interactive text worlds. Our method expands on the partial game construction task in Fan et al. (2020) to achieve full construction, including the creation of novel items and their coherent arrangement. Moreover, adopting the basic steps of constructing locations, characters, and objects as described in this work, we elaborate it with the memories of relevant and previous constructions for creativity and coherence.

Actions are the executable movements in games through which a player interacts with objects. Ammanabrolu et al. (2020a) presents an automated story plot generation method that leverages soft causal relations and commonsense reasoning via the transformer-based model COMET, constructing a plot graph from high-level plot points extracted from an input story plot. Yao et al. (2020) introduce the Contextual Action Language Model (CALM), which leverages reinforcement learning and GPT-2 to generate contextually appropriate actions in text-based games. Ammanabrolu et al. (2019) introduces a technique for automatically generating cooking quests in a "home"-themed text-adventure game using knowledge graphs and fine-tuned GPT-2. Al-Nassar et al. (2023) explores using BERT and GPT-2 models to automatically generate engaging quests for video games. Värtinen et al. (2024) investigates the procedural generation of RPG quest descriptions using GPT-2 and GPT-3. It aligns with our research on employing GPT-4 to generate a dynamic world for an interactive text-based game. These works have developed meticulous methodologies to maintain plot coherence, which serves as a complement for our work. Our work focuses on generating executable code actions in games, with detailed constraints such as preconditions and effects.

7 Discussion and Future Work

This study has demonstrated a framework for procedural textual-world generation that uses task-specific modules and few-shot prompting to create playable interactive fiction games. Our system integrates multiple LLM modules to generate game components including locations, NPCs, and actions. The outputs are synthesized into a gameplay class, which are processed by a game parser to create playable games.

Despite the incorporation of programmatic checks designed to ensure the correctness of JSON outputs, the initial output from our models continues to exhibit slight instability. To address these, we are exploring potential enhancements such as increasing the number of shots provided to each model or conducting targeted fine-tuning of the models. We are also investigating methods to reduce the latency of generation to ensuring a smoother and more engaging user experience during the iterative world-building process.

In future work, we plan to enhance the interactivity of our world-generation system by expanding our human-in-the-loop capabilities to not only allow direct edits but also integrate user feedback for dynamic content regeneration, employing techniques like fine-tuning as discussed by Patel et al. (2024). Additionally, we aim to evolve our system into a multimodal platform, inspired by the Bruce et al. (2024), which uses diverse inputs like text and images to generate interactive environments. These advancements will make our narrative generation more responsive and immersive, tailoring experiences to user interactions and broadening the scope of our work.

8 Acknowledgements

We would like to express our gratitude to the Interactive Fiction and Text Generation course at the University of Pennsylvania for providing the foundational knowledge and inspiration for this research.

Special thanks to James Dennis for his invaluable contributions in building the Text Adventure Games framework, which significantly aided our work.

We also extend our appreciation to our anonymous reviewers for their constructive feedback, which greatly improved the quality of this paper.

References

Suzan Al-Nassar, Anthonie Schaap, Michael Zwart, Mike Preuss, and Marcello Gómez-Maureira. 2023. Questville: Procedural quest generation using nlp models. pages 1–4.

Prithviraj Ammanabrolu, William Broniec, Alex Mueller, Jeremy Paul, and Mark O. Riedl. 2019. Toward automated quest generation in text-adventure games. *CoRR*, abs/1909.06283.

Prithviraj Ammanabrolu, Wesley Cheung, William Broniec, and Mark O. Riedl. 2020a. Automated storytelling via causal, commonsense plot ordering. *CoRR*, abs/2009.00829.

Prithviraj Ammanabrolu, Wesley Cheung, Dan Tu, William Broniec, and Mark O. Riedl. 2020b. Bringing stories alive: Generating interactive fiction worlds. *CoRR*, abs/2001.10161.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Jake Bruce, Michael Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Bechtle, Feryal Behbahani, Stephanie 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. *Preprint*, arXiv:2402.15391.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. 2019. Textworld: A learning environment for text-based games. *Preprint*, arXiv:1806.11532.

Paramveer S. Dhillon, Somayeh Molaei, Jiaqi Li, Maximilian Golub, Shaochun Zheng, and Lionel P. Robert. 2024. Shaping human-ai collaboration: Varied scaffolding levels in co-writing with language models. *Preprint*, arXiv:2402.11723.

Angela Fan, Jack Urbanek, Pratik Ringshia, Emily Dinan, Emma Qian, Siddharth Karamcheti, Shrimai Prabhumoye, Douwe Kiela, Tim Rocktaschel, Arthur Szlam, et al. 2020. Generating interactive worlds with text. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1693–1700.

Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. 2020. Interactive fiction games: A colossal adventure. *Preprint*, arXiv:1909.05398.

Matthew Hausknecht, Ricky Loynd, Greg Yang, Adith Swaminathan, and Jason D. Williams. 2019. Nail: A general interactive fiction agent. *Preprint*, arXiv:1902.04259.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015. Language understanding for text-based games using deep reinforcement learning. *Preprint*, arXiv:1506.08941.

Muhammad U Nasir, Steven James, and Julian Togelius. 2024. Word2world: Generating stories and worlds through large language models. *arXiv preprint arXiv:2405.06686*.

Philip Osborne, Heido Nõmm, and Andre Freitas. 2021. A survey of text games for reinforcement learning informed by natural language. *Preprint*, arXiv:2109.09478.

Zeeshan Patel, Karim El-Refai, Jonathan Pei, and Tianle Li. 2024. Swag: Storytelling with action guidance. *Preprint*, arXiv:2402.03483.

Ahmed Y Radwan, Khaled M Alasmari, Omar A Abdulbagi, and Emad A Alghamdi. 2024. Sard: A human-ai collaborative story generation. *arXiv preprint arXiv:2403.01575*.

Yuqian Sun, Zhouyi Li, Ke Fang, Chang Hee Lee, and Ali Asadipour. 2023. Language as reality: a co-creative storytelling game experience in 1001 nights using generative ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 19, pages 425–434.

Jack Urbanek, Angela Fan, Siddharth Karamcheti, Saachi Jain, Samuel Humeau, Emily Dinan, Tim Rocktäschel, Douwe Kiela, Arthur Szlam, and Jason Weston. 2019. Learning to speak and act in a fantasy text adventure game. *Preprint*, arXiv:1903.03094.

Susanna Värtinen, Perttu Hämäläinen, and Christian Guckelsberger. 2024. Generating role-playing game quests with gpt language models. *IEEE Transactions on Games*, 16(1):127–139.

Shunyu Yao, Rohan Rao, Matthew J. Hausknecht, and Karthik Narasimhan. 2020. Keep CALM and explore: Language models for action generation in text-based games. *CoRR*, abs/2010.02903.

A Background and player

Listing 1: A few-shot example for playthrough generation

```
background story:
talking cat

player:
Whisper

Whisper is a sleek, nimble, and mischievous talking cat with shiny, silver fur. While lovably
    eccentric and always ready for a playful adventure, Whisper possesses wisdom and knowledge,
    which comes through in his eloquent and captivating narratives.

starting state:
Kitchen

winning state:
Whisper reaches the top of the mountain.
```

B Location prompt and output

Listing 2: Example list of purposeful locations

```
System Prompt:
You are a helpful location generator for building a text adventure game. Given the background story
    of the game, the description of the player, the initial state and winning state of the game,
    the entire list of actions the player needs to take in order to win the game, you should output
    a list of locations that you think is needed for the player to be able to take all the
    specified actions and for the game to operate properly, based on the list of actions the user
    needs to take. Each location should be one specific place. Here is more information on what the
    starting location should be like: {central_loc_desc}. Mention which location is the starting
    location of the game in its description.

Output:
{
  "Enchanted Meadow": "Where the magic feather can be found.",
  "Cottage": "Starting location. Where the player must unlock the door and then later return to
    celebrate victory.",
  "Forest Clearing": "The player must find a path to enter the dark forest.",
  "Dark Forest": "Where the glowing mushrooms can be gathered.",
  "Ancient Well": "A mysterious location where the magic feather must be dropped to receive
    enchanted wing armor.",
  "Moonlit Sky": "Where the player must wear the enchanted wing armor and fly upwards towards the
    moon.",
  "Ghostly Cloud": "A mystical area where the player must use glowing mushrooms to repel ghosts.",
  "Moon Dancer's Labyrinth": "Where Moon Dancer is held captive by ghosts.",
  "Celestial Realm": "Where Moon Dancer resides and where the player must eventually return to
    celebrate victory."
}
```

C Character Prompt and Output

Listing 3: Character Generation Prompt and Output

```
System Prompt:
You are a helpful main character generator for building a text adventure game based on a given
    story. Generate one main character from this story in a JSON format. Remember to leave the
    values to the location and inventory keys empty. Output the character in JSON format, like
    this: {character_format}

Output JSON Example:

   "characters": [
        {
            "name": "Whisper",
        "description": "Whisper is a sleek, nimble, and mischievous talking cat with shiny,
    silver fur. While lovably eccentric and always ready for a playful adventure, Whisper possesses
    wisdom and knowledge, which comes through in his eloquent and captivating narratives.",
        "persona": "Known for his quick-witted humor, flirtatious charm and a peculiar way of
    seeing the world, Whisper loves to play devil's advocate and challenge others to think
    critically. He relies on his enchanting voice to persuade, console, and encourage his fellow
    adventurers.",
        "location": "",
        "goal": "",
        "inventory": {}
    }]
```

D NPC Prompt and Output

Listing 4: NPC Generation Prompt and Output

```
System Prompt:
"You are a helpful character generator for building a text adventure game.Your job is to generate
    NPC characters in a given location. Given the location's name, description, purpose, as well as
    the description of the player of the game from the user, you should generate a list of 3
    suitable and purposeful NPCs that should be in this location. You should espeially focus on
    aligning with the location's given purpose. Output the NPCs as a list of JSON objects. You
    should always populate name, description, persona, goal, and properties of an NPC character.
    You should always leave the location and inventory of each character empty. Here are the NPCs
    you have already before and SHOULD NOT generate again: {existing_npcs}"

Output JSON Example:

   "characters": [
       {
           "name": "Bert the Beagle",
           "description": "Bert is a kind-hearted and slightly clumsy Beagle who always keeps an
    eye on the Kitchen. Even though he's a dog, he shares a mutual respect and moderate friendship
    with Whisper due to their shared domesticated life.",
           "persona": "I'm Bert, always there for a good sniff and a wagging tail. I may be a bit
    messy, but I love it here in the Kitchen. And don't tell anyone... but I have a soft spot for
    that clever cat, Whisper.",
           "location": "",
           "goal": "To help Whisper in any way he can and maintain harmony in the kitchen.",
           "inventory": {}
            "properties": {}
       }]
```

E Item prompt and Output

Listing 5: Item Prompt and Output

```
System Prompt:
"You are a helpful items generator for building a text adventure game.Given the location name, its
    description, and its purpose from the user,generate a list of 5 suitable and purposeful items
    that should be in this location.Remember,each item should be specific and purposeful, and
    should not be a character or a location. Output the items as a list of JSON items.Include all
    necessary attributes such as name, description, examine text, and properties.Here are the items
    you have already generated for this location before and SHOULD NOT generate again:
    {existing_items}."

Output JSON Example:
   "items": {
           "Shopping Cart": {
               "name": "Shopping Cart",
               "description": "A silver metal shopping cart, it's slightly rusty but still usable.",
               "examine_text": "You see a shopping cart, it's large enough to carry a respectable
    number of items. The wheels seem to be in good condition.",
               "properties": {
                   "is_container": true,
                   "is_drink": false,
                   "is_food": false,
                   "is_gettable": false,
                   ”is_surface”: false,
                   ”is_weapon”: false,
                   ”is_wearable”: false
               },
               ”location”: ”Costco Parking Lot”
           }
       }
```

F Block Generation Result

Listing 6: An example of a generated block for a location

```
{
    ”blocks”: {
        ”out”: {
        ”_type”: ”ProduceAisleOutBlock”,
        ”connection”: ”Costco Parking Lot”,
        ”bobby_the_shopper”: ”Bobby the Shopper”,
        ”location”: ”Produce Aisle”
        }
    }
}
```

G Action Generation Prompt and Result

Listing 7: An example of a generated location with items

```
System Prompt;
Given a list of action words, analyze each word to determine its uniqueness. For actions that have
     synonyms, consult the Oxford Dictionary to verify if they are indeed synonymous.
    For each unique action, compile a list where synonymous actions are grouped together.
    Use the format ”get [alias: procure, retrieve]” to display the primary action followed by its
     synonyms in brackets, prefixed by ”alias:”.
    Ensure to include only one primary action for each group of synonyms, with all other synonymous
     actions listed as aliases.
    DO NOT ADD the following actions : Go, Get, Drop, Inventory, Give, Unlock Door, Examine or their
     synonyms.

Output example:
class Climb(actions.Action):
    ACTION_NAME = ”climb”
    ACTION_DESCRIPTION = ”Climb something”
    ACTION_ALIASES = [”jump”]

    def __init__(self, game, command: str):
        super().__init__(game)
        self.character = self.parser.get_character(command)
        self.item = self.parser.match_item(
            command,
            self.parser.get_items_in_scope(
            self.character)
        )

    def check_preconditions(self) -> bool:

        # Preconditions:
        # * There must be a matched item
        # * The item must be climbable

        if not self.was_matched(self.item):
            return False
        if not self.item.get_property(”is_climbable”):
            description = ”That's not something you can climb.”
            self.parser.fail(description)
            return False
        return True
    def apply_effects(self):

        # Effects:
        # * Describes the climbing
        description = "{name} climbs the {item}.".format(
        name=self.character.name.capitalize(), item=self.item.name
        )
        self.parser.ok(description)
```

H Final Game JSON Example

Listing 8: An example of the final game JSON generated

```
{
    "player": "Whisper",
    "start_at": "Kitchen",
    "game_history": [],
    "game_over": false,
    "game_over_description": null,
    "characters": [
        {
            "name": "Whisper",
            "description": "Whisper is a sleek, nimble, and mischievous talking cat with shiny,
     silver fur. While lovably eccentric and always ready for a playful adventure, Whisper possesses
     wisdom and knowledge, which comes through in his eloquent and captivating narratives.",
            "persona": "Known for his quick-witted humor, flirtatious charm and a peculiar way of
     seeing the world, Whisper loves to play devil's advocate and challenge others to think
     critically. He relies on his enchanting voice to persuade, console, and encourage his fellow
     adventurers.",
            "location": "Kitchen",
            "goal": "",
            "inventory": {...}
        },
        ...
        ]
        "locations": [
        {
            "name": "Kitchen",
            "description": "The kitchen is the familiar location filled with tantalizing and
     inviting smells, where our feline hero, Whisper usually spent most of its time. The daily
     hustle and bustle of human activities often left behind a tantalizing array of scents and
     morsels that tickled the curious kitty's fancy. From the window Whisper could see the entire
     suburban world, a world filled with adventures beyond anyone's imagination. This was Whisper's
     favorite perch, a place from where our stealthy adventurer embarked on countless journeys into
     the great suburban expanses.",
            "connections": {
                "east": "Backyard Tree",
                "west": "Neighbor's Garden"
            },
    "travel_descriptions": {
        "east": "Leaving the familiar scents of the kitchen, you venture towards the window, the
     gateway to the suburban wilderness, finally embarking on an adventure towards the backyard tree
     that stands tall in the eastern side, its branches offering a panoramic view of distant
     mountains.",
        "west": "Leaving the aromatic arena of the Kitchen, you make a courageous leap towards the
     wilderness, incrementally crossing the unseen boundary which plunges you into the mesmerizing
     world of the Neighbor's Garden, where a secretive conversation with the neighbouring feline
     awaits."
    },
    "blocks": {
        "east": {
            "_type": "KitchenEastBlock",
            "connection": "Backyard Tree",
            "bert_the_beagle": "Bert the Beagle",
            "location": "Kitchen"
        }
    },
    "items": {
        "Fish-Shaped Cookie": {
            "name": "Fish-Shaped Cookie",
            "description": "A cookie, lovingly shaped like a fish. Whisper's favorite treat.",
            "examine_text": "The cookie smells delicious and strongly of fish, making it irresistible to Whisper.",
            "properties": {
                "is_container": false,
                "is_drink": false,
                "is_food": true,
                "is_gettable": true,
                "is_surface": false,
                "is_weapon": false,
                "is_wearable": false
            },
            "location": "Kitchen"
        }
    },
    "characters": {
        "Bert the Beagle": {
            "name": "Bert the Beagle",
            "description": "Bert is a kind-hearted and slightly clumsy Beagle who always keeps an eye on the Kitchen. Even though he's a dog, he shares a mutual respect and moderate friendship with Whisper due to their shared domesticated life.",
            "persona": "I'm Bert, always there for a good sniff and a wagging tail. I may be a bit messy, but I love it here in the Kitchen. And don't tell anyone... but I have a soft spot for that clever cat, Whisper.",
            "location": "Kitchen",
            "goal": "To help Whisper in any way he can and maintain harmony in the kitchen.",
            "inventory": {
        "Bert's Water Dish": {
            "name": "Bert's Water Dish",
            "description": "A blue round dish kept by the family for Bert's water.",
            "examine_text": "The dish is bright blue and made from sturdy plastic. The edges are a bit chewed up but it serves its purpose well. Good for providing hydration!",
            "properties": {
                "is_container": true,
                "is_drink": true,
                "is_food": false,
                "is_gettable": false,
                "is_surface": true,
                "is_weapon": false,
                "is_wearable": false
            }
        }
    },
    "properties": {
        "character_type": "dog",
        "is_dead": false,
        "is_friendly": true
    }
},
...
"has_been_visited": false,
"commands": [],
"properties": {}
},
...
]}
```

I Baseline Model Prompt and Output

Listing 9: system prompt of baseline model

```
You are a helpful text adventure game generator. Given the background story, starting state, and winning state of the game, you should generate a game JSON in the following example format:
{
    "player": "",
    "start_at": "Example Location",
    "game_history": [],
    "game_over": false,
    "game_over_description": null,
    "characters": [
        {
            "name": "Example Character",
            "description": "",
            "persona": "",
            "location": "",
            "goal": "",
            "inventory": {...}
        },
        ...
    ],
    "locations": [
        {
            "name": "Example Location",
            "description": "",
            "background": "",
            "commands": [],
            "properties": {},
            "blocks": {},
            "travel_descriptions": {
                "east": "",
                ...
            },
            "connections": {
                "east": "",
                ...
            },
            "items": {
                "Example Location Item": {
                    "name": "Example Location Item",
                    "description": "",
                    "examine_text": "",
                    "properties": {
                        "is_container": false,
                        "is_drink": false,
                        "is_food": false,
                        "is_gettable": true,
                        "is_surface": false,
                        "is_weapon": true,
                        "is_wearable": false
                    },
                    "location": "Example Location"
                },
                ...
            },
            "characters": {
                "Example Character": {
                    "name": "Example Character",
                    "description": "",
                    "persona": "",
                    "location": "",
                    "goal": "",
                    "inventory": {
                        "Example Inventory Item": {
                            "name": "",
                            "description": "",
                            "examine_text": "",
                            "properties": {
                                "is_container": false,
                                "is_drink": false,
                                "is_food": true,
                                "is_gettable": true,
                                "is_surface": false,
                                "is_weapon": false,
                                "is_wearable": false
                            },
                            "commands": []
                        },
                        ...
                    }
                },
                ...
            },
            ”has_been_visited”: false,
        },
        ...
    ],
    ”actions”: []
}
The ... means you should expand with more items.
Be sure to populate all empty string fields. Each location should have NPC characters and items in
    it.
Each character should have objects in their inventory.
The player needs to be an existing character.
Generate at least 5 locations and 5 characters.
```

Listing 10: output of baseline model

```
{
    ”player”: ”Fluffy the Talking Cat”,
    ”start_at”: ”Kitchen”,
    ”game_history”: [],
    ”game_over”: false,
    ”game_over_description”: null,
    ”characters”: [
        {
            ”name”: ”Fluffy the Talking Cat”,
            ”description”: ”A smart looking cat with the ability to speak human language.”,
            ”persona”: ”Adventurous and curious.”,
            ”location”: ”Kitchen”,
            ”goal”: ”Reach the top of the mountain.”,
            ”inventory”: {
               ”Catnip”: {
                   ”name”: ”Catnip”,
                   ”description”: ”A herb popular among cats.”,
                   ”examine_text”: ”It's a bag of dried catnip.”,
                   ”properties”: {
                       ”is_container”: true,
                       ”is_drink”: false,
                       ”is_food”: true,
                       ”is_gettable”: true,
                       ”is_surface”: false,
                       ”is_weapon”: false,
                       ”is_wearable”: false
                   },
                   ”commands”: []
              }
            }
        },
        {
            ”name”: ”Spike the Dog”,
            ”description”: ”A friendly and playful dog.”,
            ”persona”: ”Playful and protective.”,
            ”location”: ”Living Room”,
            ”goal”: ”To have fun”,
            ”inventory”: {}
        },
        {
            ”name”: ”Oliver the Owl”,
            ”description”: ”A wise and old owl.”,
            ”persona”: ”Wise and knowledgeable.”,
            ”location”: ”Woods”,
            ”goal”: ”To impart wisdom”,
            ”inventory”: {}
        },
        {
            ”name”: ”Bobby the Bear”,
            ”description”: ”A large and intimidating but gentle bear.”,
            ”persona”: ”Calm and strong.”,
            ”location”: ”Mountain Base”,
            ”goal”: ”To protect the forest”,
            ”inventory”: {}
        },
        {
            ”name”: ”Eagle Eye”,
            ”description”: ”A mighty majestic eagle.”,
            ”persona”: ”Brave and fast.”,
            ”location”: ”Mountain Peak”,
            ”goal”: ”Keep an eye on everything”,
            ”inventory”: {}
        }
    ],
    ”locations”: [
        {
            ”name”: ”Kitchen”,
            ”description”: ”A typical home kitchen with all the appliances.”,
            ”background”: ”Home kitchen”,
            ”commands”: [],
            ”properties”: {},
            ”blocks”: {”north”: ”A wall blocks your way to the north.”},
            ”travel_descriptions”: {”west”: ”To the west is the Living Room.”},
            ”connections”: {”west”: ”Living Room”},
            ”items”: {},
            ”characters”: { ”Fluffy the Talking Cat”},
            ”has_been_visited”: true
        },
        {
            ”name”: ”Living Room”,
            ”description”: ”A cozy living room filled with comfortable furniture.”,
            ”background”: ”Home living room”,
            ”commands”: [],
            ”properties”: {},
            ”blocks”: {”east”: ”A wall blocks your way to the east.”},
            ”travel_descriptions”: {”west”: ”To the west are the Woods.”},
            ”connections”: {”west”: ”Woods”},
            ”items”: {},
            ”characters”: { ”Spike the Dog”},
            ”has_been_visited”: false
        },
        {
            ”name”: ”Woods”,
            ”description”: ”A dense, dark forest.”,
            ”background”: ”Forest”,
            ”commands”: [],
            ”properties”: {},
            ”blocks”: {},
            ”travel_descriptions”: {”north”: ”To the north is the Mountain Base.”},
            ”connections”: {”north”: ”Mountain Base”},
            ”items”: {},
            ”characters”: {”Oliver the Owl”},
            ”has_been_visited”: false
        },
        {
            ”name”: ”Mountain Base”,
            ”description”: ”The start of an intimidating rocky mountain.”,
            ”background”: ”Mountain base”,
            ”commands”: [],
            ”properties”: {},
            ”blocks”: {},
            ”travel_descriptions”: {”north”: ”Northwards leads to the Mountain Peak.”},
            ”connections”: {”north”: ”Mountain Peak”},
            ”items”: {},
            ”characters”: {”Bobby the Bear”},
            ”has_been_visited”: false
        },
        {
            ”name”: ”Mountain Peak”,
            ”description”: ”The top of the high mountain with a breathtaking view.”,
            ”background”: ”Mountain peak”,
        ”commands”: [],
        ”properties”: {},
        ”blocks”: {},
        ”travel_descriptions”: {},
        ”connections”: {},
        ”items”: {},
        ”characters”: {”Eagle Eye”},
        ”has_been_visited”: false
    }
], ”actions”: [] }
```