Toward Using ChatGPT to Generate Theme-Relevant Simulated Storyworlds

Shi Johnson-Bey[1], M.S., Michael Mateas[1], Ph.D. and Noah Wardrip-Fruin[1], Ph.D.

[1]*University of California Santa Cruz, 1156 High St., Santa Cruz, California 95064, USA*

Abstract

While simulated story worlds have proven to be fertile ground for emergent storytelling in research and in commercial game genres such as life simulation, 4X, and roguelikes, they have also proven challenging to create. Building these simulations can involve hand-authoring large amounts of content that provide context for character decision-making, ensure variation among generated narratives, and align with the simulation's narrative setting and themes. This short paper discusses preliminary work on leveraging ChatGPT to generate theme-relevant content for *Neighborly*, a character-driven settlement simulation framework designed for narrative generation. Given a textual description of the narrative setting, we demonstrate how ChatGPT can be used to generate Python code for characters' businesses, occupations, and traits. We discuss challenges with development and future work.

Keywords

ChatGPT, World Simulation, Emergent Storytelling, Mixed-initiative content generation

1. Introduction

Simulated story worlds like those seen in commercial games such as *The Sims*, *Dwarf Fortress*, *Caves of Qud*, and *Civilization* have shown to be powerful tools for emergent storytelling. They use a combination of simulated systems, autonomous characters, and procedural narrative systems to generate unique-feeling and engaging emergent experiences. However, authoring enough content to maintain the illusion of a living world is one of the medium's core challenges and is an active area of research in interactive storytelling[1]. Simulated storyworlds require a trove of hand-authored content (character types, cultures, character jobs/roles, object types, and social rules) to cultivate narrative themes and provide context for non-player character (NPC) decision-making processes.

Mixed-initiative content generation is a promising solution to relieving this authoring burden. This process involves partnering humans with computational systems to create content that neither could easily create independently [2, 3]. Usually, this collaboration involves a computational system generating potential content, providing that content to the human user for review, and refining its outputs based on the human partner's feedback. Sometimes the roles are slightly reversed as the computational system may provide feedback and insights about designs presented by the human. Research in this area has explored level generation[4], automated game design[5], and interactive storytelling[6, 7].

Large-language model (LLM) applications like ChatGPT have recently gained popularity for their versatile and unprecedented performance at various tasks, including classification, question answering, text generation, summarization, and image generation. Companies such as NVIDIA and Unity have announced new LLM-enabled technology stacks that enable game developers to create a wide array of content ranging from animations to code to autonomous virtual characters [8, 9].

This short paper discusses preliminary work on leveraging ChatGPT to generate narrative theme-relevant content for *Neighborly*, a character-driven settlement simulation framework for narrative generation[10]. Given a textual description of the narrative setting, we show how ChatGPT can be used to generate Python code for characters' businesses, occupations, and traits. Our goal is to eventually enable users to create simulations from natural language prompts. We outline the beginning stages of our generation pipeline and discuss challenges with development and future work.

2. Related work

2.1. LLMs for interactive storytelling

LLM technology became popular for interactive storytelling following the release of *AI Dungeon* in 2019. The creators trained a GPT-3 model on choose-your-own-adventure stories and used it to generate interactive fiction experiences[11]. Since then, we have seen LLMs applied to other storytelling tasks like mixed-initiative storytelling [12], AI-enabled editors for story writing[13],

AIIDE Workshop on Experimental Artificial Intelligence in Games, October 08, 2023, University of Utah, Utah, USA

ismajohn@ucsc.edu (S. Johnson-Bey); mmateas@ucsc.edu (M. Mateas); nwardrip@ucsc.edu (N. Wardrip-Fruin)

https://shijbey.github.io/ (S. Johnson-Bey)

interactive story generation[14], and dialog generation for NPCs in table-top role-playing games [15].

2.2. Generative AI in the Games industry

Companies within the commercial games industry are also using LLMs to power new generative AI tools aimed at game developers. Unity recently released *Unity Muse*, a tool to improve developer productivity by allowing users to generate assets (textures, models, animations, etc.) using a natural language interface[8]. Another example is *NVIDIA ACE for Games*, a suite of tools that leverage LLMs and other AI technologies to help game developers create hyperrealistic virtual characters in their games[9].

2.3. ChatGPT

ChatGPT is an interactive LLM application within the family of generative pre-trained transformer models released by OpenAI. Its main user interface resembles a messaging application, and users can send natural language prompts to the model. It was trained using reinforcement learning techniques [16], allowing it to respond fluently to input from the user. One of its greatest strengths is its ability to reason about semantic relationships.

2.4. ChatGPT for simulated storyworlds

ChatGPT has started to garner some attention as a potential tool for storytelling with simulated story worlds. Méndez and Gervás explored using ChatGPT for story sifting[17] – searching for interesting narrative material within a repository of simulated story world data [18]. They found that ChatGPT returned proper prose and performed well at summarization tasks. However, it often embellished the summaries, and the authors found it challenging to influence/bias ChatGPT toward desired stories and narrative themes.

Park et al. (2023) [19] used ChatGPT to power all simulation aspects, including character behavior and the behavior/state of inanimate objects like toasters and refrigerators. The core of their system was the *generative agents* architecture that used ChatGPT to handle how characters observe, remember, plan, reflect, and act upon their environment. This project showcased the potential breadth of applications for ChatGPT within simulated storyworlds, especially their potential as their aptitude for common sense reasoning and understanding of semantic relationships. However, the main drawbacks of their approach were (1) the system was not playable due to long runtimes and a lack of resource/progression tracking and (2) the lack of an authoring interface that integrated with existing game development processes. ChatGPT does not keep track of values well. Ideally, these things should be handled by simulated systems implemented using a general-purpose programming language.

3. Generating simulation content with ChatGPT

This section describes our preliminary work using chatGPT to generate content for simulated story worlds. We want to explore how LLMs, like ChatGPT, can support creators during the rapid prototyping and iteration phase of simulation development. When brainstorming content, designers might spend time consulting multiple references for inspiration. Generating initial content gives designers a starting point to jumpstart the rest of their design process. Our goals with this project are to:

- Generate content that fits the narrative setting of the story world.
- Output editable source code that integrates with the existing development ecosystem.
- Evaluate if this improves the prototyping workflow for simulation designers.

Within this short paper, we only cover the first of our three goals. The evaluation is saved for a future full-length publication. We are working on turning our generation procedure into a single cohesive software tool. A fully-featured version of this tool might operate like a virtual pair programmer that provides suggestions and feedback during the development process to help ensure user meet their authorial goals.

Thus far, our tool generates character spawn information, business types, occupation types, and character traits for *Neighborly*, a character-driven simulation framework for narrative generation [10]. *Neighborly* simulates the lives and relationships of generations of characters in a generated settlement. Characters are born, grow older, work jobs, form relationships, have families, and engage in a myriad of life events. We chose *Neighborly* because it is Python-based and allows us to load custom simulation data using plugins. We chose to use ChatGPT because, without any fine-tuning, it performs well at outputting structured information and code. We wanted to focus more on developing the data generation pipeline. So, rather than fine-tune a GPT-2 model on Neighborly-specific APIs, we chose to ask ChatGPT to fill JSON templates that we could post-process into Python source code. Also, this allowed us to explore what domain knowledge we could leverage from ChatGPT without additional tuning.

Content generation starts with a natural language description of the narrative setting of the simulation. For instance, “A cyberpunk city”. This initial prompt is then used inside a collection of follow-up prompts. We asked

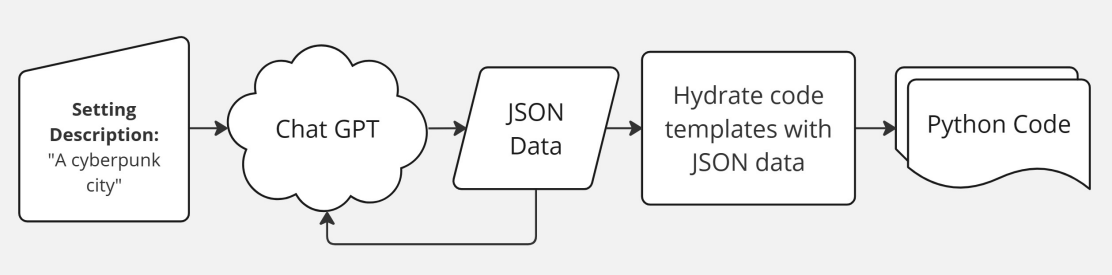

Figure 1: A flow chart of the data generation pipeline. ChatGPT may be used multiple times to generate additional data that depends on ChatGPT's previous response. The final Python output is saved to files for later use.

that ChatGPT output its responses as structured JSON. This data format allowed us to post-process intermediate results and send follow-up requests to ChatGPT for additional information. Once we have all the required information, we generate Python files by hydrating Jinja2[1] templates using the combined JSON output from ChatGPT. The final product is valid Python source code that can be edited in a text editor and packaged inside a *Neighborly* plugin for later use. Figure 3.3 shows the existing data generation pipeline. If users have conflicts with or want to adjust the content generated by the pipeline, they can refine their initial prompt or edit the produced Python code.

We focused on the three major content areas for theme-building in Neighborly: businesses and occupations, characters, and character traits. Based on what Neighborly's APIs provide, these were the most straightforward to generate content for since they rely on static configuration data. They are not directly responsible for narrative generation, but they affect the growth of character relationships and support the setting of the simulated story world. Generating life events for characters would have had a more direct impact on narrative generation. However, Neighborly does not have a declarative way to define event types and their preconditions and post effects.

3.1. Generating businesses and occupations

First, we attempted to generate new business types for characters to own and work at. In Neighborly, these set the tone for which characters interact and provide additional context to storytelling. Generating new types was a two-step process as business types define occupations that may need to be generated. After ChatGPT provides business types in the format provided in the listing below, we scrape the owner and employee types and re-query ChatGPT configuration for those occupations. When all data has been collected, it is passed to Jinja to create appropriate Python source code files. We did not have any issues with this generation phase and were surprised by ChatGPT's ability to create arbitrary numbers, occupation names, and business services (See Listing 1).

Listing 1: The JSON output provided by ChatGPT for a business that might exist in a cyber punk world. (See prompt listing 2).

```
[
    {
        "name": "Cybernetic
            Augmentation Clinic
        "owner_type": "Cybernetic
            Surgeon",
        "employee_types": {
            "Cybernetic Technician":
                4,
            "Administrative Assistant
                ": 2,
            "Receptionist": 1,
        },
        "services": ["Cybernetic
            implants", "Neural
            Enhancements", "
            Augmentation consultations
            "],
    },
    ...
]
```

3.2. Generating characters

Next, we tried generating configurations for the types of characters that spawn into the simulation. We found that ChatGPT had trouble with this task. It likes to generate the typical fantasy races (Humans, Elves, Orcs, etc.) found in role-playing games, even if the setting is not entirely appropriate for those categories. Also, it has

[1] https://jinja.palletsprojects.com/en/3.0.x/

trouble differentiating between character types and roles. For example, in response to the prompt, "List races of characters that exist within a cyberpunk futuristic story world?", we received output that included humans, cyborgs, and androids, but also definitions for organizations and roles such as "street gangs", "company executives", and "Fixers".

Modifying the phrasing also did not help as expected. Swapping "race" for character type resulted in a list of narrative-inspired roles like *hero*, *villain*, and *mentor*. Furthermore, swapping "race" for species resulted in ChatGPT generating entirely made-up fantasy-style species like "Ferbles", "Aquamids", and "Plantlings" which did not fit the theme.

One solution to this problem was changing ChatGPT's task from generating character types to determining which types from a specified list are most appropriate for appearing in the given setting and relative spawn frequencies based on the other entries.

3.3. Generating character traits

The last content type we experimented with was character traits. *Crusader Kings* extensively uses traits to drive character relationship development and other mechanics. However, creating a list of traits and determining their effects relative to each other is time-consuming manual work. We offloaded this to ChatGPT and had it provide JSON feedback that included trait names, descriptions, and social rules for modifying the platonic and romantic compatibility of two given characters (see Figure 3.3).

```
{
    "name": "Optimistic",
    "description": "This character is positive and hopeful.",
    "incompatible_with": "Pessimistic",
    "relationship_modifiers": [
        {
            "has_trait": "Optimistic",
            "romantic_compatibility": 0,
            "platonic_compatibility": 4
        },
        {
            "has_trait": "Pessimistic",
            "romantic_compatibility": -2,
            "platonic_compatibility": -1
        },
        {
            "has_trait": "Cynical",
            "romantic_compatibility": -3,
            "platonic_compatibility": -2
        },
        {
            "has_trait": "Skeptical",
            "romantic_compatibility": -1,
            "platonic_compatibility": 0
        }
    ]
}
```

Figure 2: An "Optimistic" trait generated by ChatGPT with a description and other metadata to be fed into the templating engine to produce Python code. (See prompt listing 4).

4. Discussion

Overall, ChatGPT performs well at generating business and occupation definitions. Occasionally, it will misinterpret a configuration parameter. However, this can be easily corrected by the human author. Also, validation safeguards could be put in place during content generation to notify the user of any errors before final code generation.

Even with the occasional error, generating content this way was still much faster than authoring all the definitions by hand. ChatGPT really shines when generating fuzzy values for configuration settings such as the relative socioeconomic status of occupations, the lifespan of businesses, and the relative numbers of employee types. It allows authors to quickly reach the point of having a running simulation, allowing them to tweak the values at a later time.

ChatGPT's ability to create arbitrary service types for business was surprising and feels like it adds to building up the theme of a Cyberpunk world. However, since AI systems use the list of services for character decision-making, arbitrary string values can easily become an authoring nightmare. Adding 10 new business types could yield 10 to 30+ new service types that they would need to author new simulation rules. If we constrain the list of services to be selected from a fixed set, then we could create new business types that leverage existing rules, further simplifying content authoring.

This work needs to be evaluated in a user study. Tentatively, the plan is to present participants with two authoring tasks. The first task would ask participants to hand-author a subset of simulation content, and the second would ask them to generate it with the tool and modify the results. We believe that users will find the ChatGPT-powered workflow takes significantly less time.

5. Conclusion

This short paper presents preliminary work on using ChatGPT to generate theme-relevant content for an agent-based social simulation. We aimed to simplify the content prototyping process by leveraging ChatGPT's semantic knowledge to generate Python code. We describe our pipeline for generating characters, businesses, and character traits from a short description of a designer's narrative setting/theme. We are working on turning our generation pipeline into a single tool to facilitate user

studies. As future work, we would like to explore using an LLM to generate the storylet-style events that Neighborly uses for narrative generation.

References

[1] C. Hargood, D. E. Millard, A. Mitchell, U. Spierling, The authoring problem: An introduction, in: The Authoring Problem: Challenges in Supporting Authoring for Interactive Digital Narratives, Springer, 2023, pp. 1–13.

[2] G. N. Yannakakis, A. Liapis, C. Alexopoulos, Mixed-initiative co-creativity, Foundations of Digital Games (2014).

[3] A. Liapis, G. Smith, N. Shaker, Mixed-initiative content creation, Procedural content generation in games (2016) 195–214.

[4] G. Smith, J. Whitehead, M. Mateas, Tanagra: A mixed-initiative level design tool, in: Proceedings of the Fifth International Conference on the Foundations of Digital Games, 2010, pp. 209–216.

[5] S. Cardinale, M. Cook, S. Colton, Ai-driven sonification of automatically designed games (2022).

[6] M. Kreminski, M. Dickinson, N. Wardrip-Fruin, M. Mateas, Loose ends: a mixed-initiative creative interface for playful storytelling, in: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, volume 18, 2022, pp. 120–128.

[7] A. Alvarez, J. Font, J. Togelius, Story designer: Towards a mixed-initiative tool to create narrative structures, in: Proceedings of the 17th International Conference on the Foundations of Digital Games, 2022, pp. 1–9.

[8] M. Whitten, Introducing unity muse and unity sentis, ai-powered creativity, Unity Blog (2023). URL: https://blog.unity.com/engine-platform/introducing-unity-muse-and-unity-sentis-ai.

[9] A. Burnes, Introducing nvidia ace for games - spark life into virtual characters with generative ai, NVIDIA GeForce (2023). URL: https://www.nvidia.com/en-us/geforce/news/nvidia-ace-for-games-generative-ai-npcs/.

[10] S. Johnson-Bey, M. J. Nelson, M. Mateas, Neighborly: A sandbox for simulation-based emergent narrative, in: 2022 IEEE Conference on Games (CoG), IEEE, 2022, pp. 425–432.

[11] Latitude, Ai dungeon, 2019.

[12] E. Nichols, L. Gao, R. Gomez, Collaborative storytelling with large-scale neural language models, in: Proceedings of the 13th ACM SIGGRAPH Conference on Motion, Interaction and Games, 2020, pp. 1–10.

[13] A. Yuan, A. Coenen, E. Reif, D. Ippolito, Wordcraft: story writing with large language models, in: 27th International Conference on Intelligent User Interfaces, 2022, pp. 841–852.

[14] J. Freiknecht, W. Effelsberg, Procedural generation of interactive stories using language models, in: Proceedings of the 15th International Conference on the Foundations of Digital Games, 2020, pp. 1–8.

[15] J. Kelly, M. Mateas, N. Wardrip-Fruin, Towards computational support with language models for ttrpg game masters, in: Proceedings of the 18th International Conference on the Foundations of Digital Games, 2023, pp. 1–4.

[16] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, G. Irving, Fine-tuning language models from human preferences, arXiv preprint arXiv:1909.08593 (2019).

[17] G. Méndez, P. Gervás, Using chatgpt for story sifting in narrative generation, International Conference on Computational Creativity (2023).

[18] J. Ryan, Curating simulated storyworlds, University of California, Santa Cruz, 2018.

[19] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, M. S. Bernstein, Generative agents: Interactive simulacra of human behavior, arXiv preprint arXiv:2304.03442 (2023).

A. Appendix

Listing 2: Prompt for eliciting business types.

```
List the types of businesses that may
    exist in a CyberPunk city, what
    services they offer (as a list of
    single words), the job title of the
    owner, the job titles of employees,
    and the number of employees. Use the
    following YAML template:

name: <business type>
components:
    Name:
      value: <business name>
    Business:
      owner_type: <owner job title>
      employee_types:
        <employee title>: <quantity>
        <employee title>:  <quantity>
        ...
    Services:
      services: <service, service, ...>
```

Listing 3: Prompt for eliciting occupation status levels.

```
List the types of occupations that exist
    in a CyberPunk city and their
    relative social status on a scale
    from 1 to 5 with 5 being the highest.

Use the following template:
[
    {
        "name": "<occupation name>"
        "social status": <social status>
    },
    ...
]
```

Listing 4: Prompt for eliciting character trait types.

```
Generate a list of character traits for
    NPCs in a simulation. The list should
     include the name of the trait, a
    short description, and a list of
    other traits that it is incompatible
    with. Each entry should also include
    a list of relationship modifiers that
     list the trait of an interlocutor
    and corresponding romantic and
    platonic compatibility modifiers on a
     scale [-5, 5]. Return the output as
    valid JSON.

Example:
[
    {
        "name": "Friendly",
        "description": "This character
            naturally gets along with
            others,
        "incompatible_with": ["Mean",
            ...],
        "relationship_modifiers": [
            {
                "has_trait": "Friendly",
                "romantic_compatibility:
                    0,
                "platonic_compatibility:
                    3,
            },
            ...
        ]
    },
    ...
]
```