Pseudo Lexical Generation - Generating Small Narrative Languages

Harris Sinclair, Alex Broadaway, Jon Urrutia

1 Introduction

Language is one of the most versatile tools humanity has ever developed, capable of encoding complex ideas, emotions, and stories. In computational linguistics, there is a growing interest in not only understanding and processing existing languages but also in generating entirely new ones. This process, often referred to as "pseudo lexical generation," explores how small narrative languages—synthetic languages with specific purposes or constrained vocabularies—can be created to support storytelling, world-building, and computational experiments.

Small narrative languages, as a subset of constructed languages (conlangs), focus on their applicability within limited contexts, such as fictional worlds, constrained storytelling environments, or machine learning applications. Unlike general-purpose languages, these synthetic constructs are often designed with targeted communicative goals in mind, such as expressing a specific cultural perspective, encoding unique storytelling mechanics, or representing abstract concepts through compact and symbolic lexicons.

The pseudo lexical generator discussed in this paper is implemented using the Godot engine. It allows users to serialize basic concepts, such as verbs, living entities, and places, into a structured dictionary. Additional words are then described using these primary concepts. The system generates random draws for the concepts, which are subsequently used to create new language symbols representing the words derived from them. This approach enables the creation of novel languages, where the limits on the number of languages generated reside solely in the set of symbols provided by the user. Each unique set of symbols defines a new language, offering an expansive potential for linguistic variation.

The applications of this system are diverse and compelling. For instance, it can be used to immerse players in the story of a game by giving "local" NPCs their own language, enhancing the depth of the fictional world. Alternatively, it could serve as the foundation for a game mechanic centered around discovering and deciphering symbols of an unknown language. These examples highlight the system's potential to enrich interactive storytelling and create unique player experiences.

This paper explores the theoretical underpinnings, methodologies, and applications of pseudo lexical generation in the context of narrative construction. It aims to establish a framework for understanding how small narrative languages can enhance storytelling, offer insights into linguistic evolution, and serve as tools for computational creativity. Through this investigation, we seek to address key questions: How can pseudo lexical generation be systematically approached? What are the practical benefits of using such languages in constrained narratives? And how can emerging technologies make this process more accessible to creators and researchers alike?

2 Investigation of the Existing

Setting out with the specific goal of generating small narrative pseudo-languages, we knew we had to first understand what makes these languages feel authentic and believable. What characteristics do they possess that allow us to perceive them as "real" in the context of a fictional world? To answer these questions, we decided to investigate pre-existing handcrafted narrative languages to uncover insights into their structure and design. By studying these languages, we hoped to identify key elements that we could incorporate into our own language generator to achieve a similar sense of authenticity.

Our journey took an interesting turn when we came across a post on Steam by a user with the alias "Aeith," who had explored the indie title Chants of Sennaar in great detail. In this post, Aeith discussed their efforts to extend and expand the languages used within the game. As they delved deeper into the game's linguistic structure, they made a number of fascinating observations, one of which would later form the foundation of our own language generation system. This discovery was pivotal to our project, as it offered a concrete example of how a small, fictional language could feel coherent and structured without relying on the complex intricacies of a full-fledged constructed language. Aeith pointed out specific glyphs within the language of the "Priests" faction that were consistently repeated, each serving as a key marker for certain concepts.
Take a look at this example from the Priests' language:

(to) Open	Me/I	Death/Dead	Abbey	Garden	Instrument
Door/Gate	(to) Go (Through)/ (to) Pass	Devotee	(to) Fin/(to) Reach	(to) Love/(to) Like	(to) Make/(to) Create
(to) Close	God	(to) Search/(to) Seek/(to) Want	(to be) Free/ Liberty	Church	Potion
Salute/Greeting(s)/ Hello/Bye	(to) Speak/(to) Tell/(to) Say	High/Up	(to) Look/(to) See	Cemetery	Lens
You	Human/Person	Warrior	Key	Music	
(to) Help/(to) Aid somebody	Plant	Not	Preacher	Vase/Pot	

By analyzing the language, Aeith discovered that certain symbols were used to represent specific themes or categories. One such glyph, for example, consistently indicated the concept of a place. This repetition of the glyph across different words helped establish a semantic link between the word and its meaning, giving the language a sense of structure and predictability.

Similarly, another glyph was used to represent living things. This use of consistent, recurring symbols to signify particular types of concepts or entities became a crucial insight for us. It was this very concept—repeated, meaningful glyphs—that would serve as the cornerstone of our language generator.

3 Building the PCG Pipeline

With our newfound understanding of the system we were building, we organized ourselves into three separate tasks, each focusing on a critical component of the project. One team member was tasked with constructing the dictionary system, which would manage and track the words within the language and their associated traits. Another team member focused on the glyph generator, responsible for creating the basic symbols that would later be used in our language. The third member worked on the compositor, a system designed to take the words and their traits from the dictionary and combine the generated glyphs to form complete language symbols.

To expedite the development process and speed up iteration times, we decided to use the free and open-source Godot Game Engine. We chose Godot due to its permissive license, which allowed us the flexibility to adapt and distribute our work freely, and its ease of use, which helped us focus more on the project itself rather than dealing with complex engine configurations or licensing issues.

The dictionary system, while seemingly simple, is a crucial part of the project. It primarily functions as a dictionary data structure, mapping strings (the words) to arrays of strings (the associated traits). However, its functionality extends beyond just storing data—it's also responsible for saving the dictionary in a JSON file. This feature ensures that our data persists across sessions, allowing the project to maintain its state and enabling us to pick up right where we left off in future work.

Next comes the more complex task of glyph generation. The glyph generator is responsible for creating the visual symbols that represent the words in our language. To ensure the glyphs are neither too simple nor too complex, we used Gaussian Randomness. This approach helps avoid the creation of unrealistically small or intricate glyphs, maintaining a balance between detail and clarity. Additionally, Gaussian Randomness is used in a more subtle way to prevent the generation of overly large or simplistic glyphs. Strokes play a key role in maintaining the integrity of the glyphs, preventing them from devolving into disconnected lines. Each stroke segment follows a Gaussian-defined length, as we discussed earlier, and its direction is determined by a uniform random distribution.

There are specific "spikes" in the distribution that increase the likelihood of the stroke aligning with one of the cardinal or intermediate directions. For instance, each cardinal direction has a 1/6 chance, while each intermediate direction has a 1/12 chance. The remaining directions are evenly distributed across the remaining 1/2 of the probability space. The decision to finish a stroke and begin a new one is made using deck randomness, a technique that not only controls the number of strokes in each glyph but also keeps the number of stroke segments varied and unpredictable.

Finally, the compositor system brings everything together. It starts by generating a mapping of trait words to their corresponding unique glyphs, as well as a similar mapping for non-trait words. After this, the compositor generates random placements for the trait glyphs. These glyph placements are randomized only once to ensure consistency throughout the language. Their positioning and recurrence are critical because repetition helps users recognize the traits associated with the words. Non-trait words are then placed at the center of the glyph, while the trait glyphs are superimposed over the main word symbol. This process creates the final symbol that will represent a complete word in our newly constructed language. The compositor's role is essential, as it takes the foundation built by the dictionary and glyph generation systems and combines them into the visual language symbols needed to communicate effectively with
the end user.

4   Example of Generation

Let us now walk through a detailed example to better understand the process. For this example, we will use a limited dictionary and walk through how we would generate a new language for the words and their associated traits.

Suppose our dictionary contains the following set of words, each with its corresponding traits:
- Man: Living, Good
- Monster: Living, Evil
- Temple: Place, Good
- Sword: Object
- Not: (No Traits)

In this case, we have five words in our dictionary: Man, Monster, Temple, Sword, and Not. Alongside the words, we also have a list of traits that describe different concepts. These traits include Living, Good, Evil, Place, and Object. Our task is to generate symbols for these words based on their associated traits.

The first step in our process is to load each of these traits into our dictionary. The dictionary is a fundamental part of the system as it helps us organize and manage the relationships between words and their traits. Once the traits are in place, we then load each word into the dictionary, including their trait associations.

This ensures that every word is connected to the correct traits, making it easier to generate appropriate glyphs later on. For instance, we know that "Man" is associated with the traits "Living" and "Good," while "Monster" is associated with "Living" and "Evil." This trait-based structure forms the foundation for the glyph generation process.

Once the dictionary is set up, the next step is to generate glyphs for each of the traits. Each trait is represented by a unique symbol that captures the essence of that concept. These glyphs are key to the language's structure, as they will be used repeatedly to form words and phrases. The symbols for the traits are carefully designed, taking into account the meaning of each trait and how they might interact visually with other symbols. For example, the glyph for "Living" might be different from the glyph for "Place," as they represent distinct concepts. These glyphs are then cataloged in a table for easy reference. Table 1 below shows the glyphs for each of the traits.

Table 1	
Living	$>$
Good	$=$
Evil	
Place	$\triangle$
Object	$+$

Now that we have generated symbols for the traits, we move on to the next step: creating symbols for the non-trait words. Non-trait words, such as "Man," "Monster," "Temple," and "Sword," do not directly correspond to a specific trait, but they are still important to the language. These words are represented by unique glyphs that incorporate the traits they are associated with. For example, the word "Man," which is associated with the traits "Living" and "Good," will combine the glyphs for these traits into a single symbol. Similarly, "Monster," associated with "Living" and "Evil," will combine the glyphs for those traits. This step ensures that each word in the language is visually linked to its meaning. The symbols for the non-trait words are presented in Table 2 below.

Table 2	
Man	
Monster	
Temple	
Sword	
Not	

With the trait glyphs and non-trait word glyphs ready, we move on to the final step: combining the glyphs into a complete symbol for each word. This is where the compositor comes in. The compositor takes the glyphs for each word, arranges them based on their trait associations, and generates the final visual representation of the word. The compositor ensures that the correct glyphs are combined in a way that makes sense both visually and semantically. For example, the word "Man" will have the glyphs for "Living" and "Good" arranged in such a way that reflects both the meaning of the word and the traits it represents. Once all the glyphs are combined, we have the final symbols ready for use. Table 3 below illustrates the final symbols for each word.

Table 3	
Man + Living + Good	
Monster + Living + Evil	
Temple + Place + Good	
Sword + Object	
Not	

By following this process, we are able to create a coherent, structured language with meaningful visual symbols for each word. The key to this process is the use of traits and their associated glyphs, which provide a consistent framework for generating symbols. Through this approach, we can generate new languages that feel both functional and authentic, allowing for creative expression while maintaining a sense of structure.

Pseudo Lexical Generation (PLG) enables a highly innovative and efficient approach to constructing languages through the use of procedural generation techniques. This method combines words with trait-based symbols, allowing for the creation of small, yet internally consistent, languages that can serve a variety of narrative and world-building needs within fictional contexts. Unlike traditional language creation, which often requires extensive resources and manual effort, PLG offers a more automated and scalable solution, making the process of language generation both faster and more flexible.

The process of constructing such a system involves multiple stages, including the definition of core traits, the creation of symbols that represent these traits, and the rules governing how these symbols interact. By using procedural techniques, PLG allows for the generation of language systems that are not only coherent and functional but also rich in variety and nuance. This approach is particularly valuable in contexts where languages need to be tailored to specific needs, such as in games, interactive fiction, or other forms of media where unique linguistic systems are required for world-building or character development.

Throughout this paper, we have demonstrated various examples of similar systems and their successful application in both computational experiments and narrative-driven contexts. These examples provide tangible evidence of the potential of PLG to streamline the language creation process while enhancing the storytelling experience. By offering a means of generating languages that feel authentic and meaningful, PLG holds significant promise for creators who wish to add layers of complexity to their fictional worlds without getting bogged down by the often arduous task of manually crafting a fully developed language.

Furthermore, we have shown how PLG can serve as a powerful tool to enhance creative storytelling, allowing writers, game designers, and other creators to generate unique languages that contribute to the immersion and depth of their work. By providing a flexible framework that can be customized to fit specific narrative needs, PLG enables creators to experiment with new linguistic concepts, making it easier to invent languages that reflect the culture, values, and worldviews of the characters and societies they are building. This paper clearly indicates the potential value of PLG as a tool for expanding the creative possibilities of language design and enhancing the overall impact of narrative experiences in interactive media and beyond.

9 References

9.1 Online References

Aeith. "Guide :: [Spoilers] Chants of Sennaar Languages." Steam Community, Valve, 17 Sept. 2023, steamcommunity.com/sharedfiles/filedetails/?id=3037327250.