

# (Genetically) Improving Novelty in Procedural Story Generation

Erik M. Fredericks  
School of Computing  
Grand Valley State University  
Allendale, USA  
frederer@gvsu.edu

Byron DeVries  
School of Computing  
Grand Valley State University  
Allendale, USA  
devrieby@gvsu.edu

**Abstract**—Procedural story generation (PCG) tailors a unique narrative experience for a player and can be accomplished via multiple techniques, from matching storylets to grammar-based generation. There exists a rich opportunity for evolutionary algorithms to be applied to this domain for intelligently constructing game narratives. This paper describes a conceptual procedure for applying genetic improvement to a grammar-driven procedural narrative within the context of a browser-based game.

**Index Terms**—novelty search, procedural story generation, genetic improvement, grammatical evolution

## I. INTRODUCTION

Procedural content generation typically focuses on developing unique experiences commonly found in genres such as roguelikes (e.g., Nethack, Dungeon Crawl Stone Soup, etc.<sup>1</sup>), where intelligent algorithms leverage a set of rules and/or random chance to instantiate varying situations. Such algorithms can lead to *emergent behaviors* that can enable a rich player experience, where such behaviors are not necessarily hard-coded by the developers.

Story generation tends to be a difficult aspect of procedural content generation, as simply combining snippets or paragraphs does not generally lead to a cohesive narrative. Narratives can be automatically crafted using trees or grammars with storylets (i.e., small pieces of narrative) that are “known” to work well together. Consider the following storylet: “You enter a new room. It is filled with lush vegetation and is quite humid. A large crustacean lounges in the corner.” Each sentence within the storylet can be procedurally generated using a large bank of phrases with meta-data to enable precise matching. For instance, the storylet can be represented as a grammar, where words surrounded by # represent future production rules:

```
S  $\mapsto$  #newRoom#. It is #verb#.ed with  
#adjective# #noun#. #randomOccurrence#.
```

Additional meta-data checks would be required to ensure that a *flow* exists as well to avoid confusing the player (e.g., it would not make sense for lush vegetation to exist within a snowy environment). This paper presents early efforts towards applying genetic improvement to procedural story narrative. Specifically, we focus on *diversifying* the set of

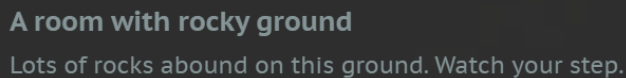
possible cohesive narrative states generated via grammar-based narrative storytelling. We next describe methods for procedural storytelling, our work in progress as an illustrative example, how we will apply genetic improvement to our story grammar, and summarize with a discussion.

## II. PROCEDURAL STORYTELLING

There exist many frameworks for enabling narrative-driven games, including Twine and Ink.<sup>2</sup> Each are open source and provide an easy-to-use interface for creating text-driven games. Games can be published as webpages or incorporated as modules in more advanced engines (e.g., Unity, Godot, Unreal Engine, etc.). Mason *et al.* recently introduced Lume, a tool for enabling procedural narrative generation via parameterized node trees that use bindings to maximize narrative coherency [1]. Tracery is an open-source tool for creating generative text using a grammar-based system that has been used in multiple applications, including procedural narratives, Twitter bots, and role-playing game mechanics [2]. Tracery has been ported to most common languages as well.

## III. WORK IN PROGRESS

Figures 1 and 2 present a sample of our procedurally-generated content. Figure 1 demonstrates a room title and description that has been generated in accordance with Simplex noise [3] and a Tracery grammar [2], and Figure 2 illustrates the space around the player, where the emoji faces represent a player and non-player characters and other characters represent Simplex noise-generated environmental features (e.g.,  $\Delta$  is a tight tunnel and  $\sim$  is a stream).



A room with rocky ground  
Lots of rocks abound on this ground. Watch your step.

Fig. 1: Room title and description.

Simplex noise is a technique commonly used for generating “smooth” environmental features in game maps [4]. For this work we are populating a two-dimensional grid with Simplex noise values, where each value translates to a specific environment feature. For instance, a noise value within [0.35, 0.55]

This work has been supported by Grand Valley State University.

<sup>1</sup>See <https://www.nethack.org/> and <http://crawl.develz.org/>, respectively.

<sup>2</sup>See <http://twinery.org/> and <https://www.inklestudios.com/ink/>, respectively.

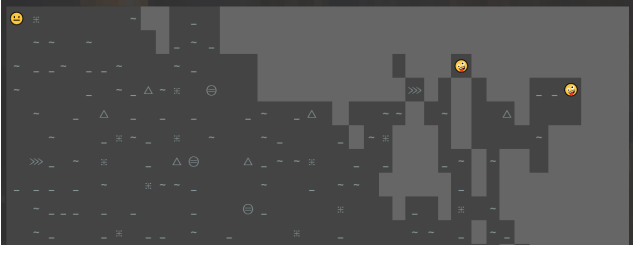


Fig. 2: Cropped minimap of environment.

results in a room with the meta-attribute `STREAM` to denote that a stream flows within the room, and we execute Tracery to generate a relevant storylet for a `STREAM`-tagged grammar. We normalize our Simplex values between  $[0.0, 1.0]$ .

A sample Tracery grammar and sentence generation for a `STREAM` environment is as follows, where the grammar has been significantly reduced in scope for presentation purposes:

```
var rules = {
  'origin': [ 'myPlace:# path#'#line #' ],
  'path': [ 'path', 'rock', 'cavern wall',
  'line': [ '#stream.a.capitalize#'],
  'nearby': [ 'beyond the #path#',
    'far away', ... ],
  'substance': [ 'light', 'reflections',
    'mist', 'shadow', 'darkness',
    'underfoot', 'stalagmites', ... ],
  'stream': [ '#stream-type#
    #stream-verb.s# #nearby#'],
  ...
};
var g = tracery.createGrammar(rules);
console.log(g.flatten('#origin#'));
```

In this example, `rules` represents the Tracery grammar, `g` is the Tracery object, and `#origin#` represents the starting point of the grammar for Tracery to parse and generate a sentence. Based on a set of grammars as illustrated, a large number of storylets can be generated via Tracery and its grammar constraints. We next describe how novelty search will be applied as a genetic improvement technique for this case study.

#### A. Genetic Improvement

A major focus with this game environment is to provide as many diverse, yet cohesive, storytelling opportunities to the player as possible. Therefore, we augment our set of Tracery grammars with grammatical evolution [5], where our fitness is replaced by novelty search [6]. Grammatical evolution focuses on evolving an individual (typically a program) via a grammar as opposed to a tree. Novelty search is an evolutionary computation-based technique for finding as many diverse, yet still optimal, solutions to a given problem. In general, novelty search uses similar evolutionary operations to genetic algorithms/grammatical evolution, however the fitness function is typically superceded by a novelty metric.

The *novelty metric* (i.e., a mathematical formula for encouraging diversity) is used within novelty search to guide

the search process to distinct areas of the solution space. For this application, we construct our novelty metric to fulfill the constraints of our Simplex noise-generated map while encouraging diversity via natural language processing (NLP) metrics. While many advanced NLP algorithms exist for calculating differences between sentences (where a study of such algorithms can form the basis of future work) [7], we will use *Word2Vec* for measuring similarity between sentences [8]. We anticipate training *Word2Vec* on open-source corpora. Equation 1 demonstrates usage of *Word2Vec* for calculating sentence similarity (following training).

$$sim(\mu_i, \mu_j) = word2vec.model(\mu_i, \mu_j), \quad (1)$$

where  $\mu_i$  and  $\mu_j$  represent sentences generated from Tracery grammars. Equation 2 next shows our novelty score calculation:

$$novelty(k) = \frac{1}{k} \sum_{i=0, j=0, i \neq j}^k sim(\mu_i, \mu_j) \quad (2)$$

The most diverse solutions (i.e., passing a novelty threshold score) are added to a *novelty archive* that is maintained throughout the course of the search procedure, where this archive is returned as output following search completion. Additional tags may be necessary to ensure that generated solutions are *feasible* from a grammatical perspective.

#### IV. DISCUSSION

This paper has introduced an early proof-of-concept for automatically improving novelty in procedurally-generated storylines for games. The proof-of-concept uses Twine to prototype the game environment, Tracery for enabling grammar-based storylet generation, and Simplex noise for generating a cohesive environment. We plan to further augment this prototype with a novelty-search based grammatical evolution heuristic for generating diverse Tracery grammars and *Word2Vec* to measure similarity.

#### REFERENCES

- [1] S. Mason, C. Stagg, and N. Wardrip-Fruin, "Lume: a system for procedural story generation," in *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 2019, pp. 1–9.
- [2] K. Compton, B. Kybartas, and M. Mateas, "Tracery: an author-focused generative text tool," in *International Conference on Interactive Digital Storytelling*. Springer, 2015, pp. 154–161.
- [3] K. Perlin, "Noise hardware. in real-time shading," *SIGGRAPH Course Notes*, 2001.
- [4] A. Patel. Making maps with noise functions. [Online]. Available: <https://www.redblobgames.com/maps/terrain-from-noise/>
- [5] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.
- [6] J. Lehman and K. O. Stanley, "Exploiting open-endedness to solve problems through the search for novelty," in *Proceedings of the Eleventh International Conference on Artificial Life*, ser. ALIFE XI. MIT Press, 2004.
- [7] S. Al-Saqqa and A. Awajan, "The use of word2vec model in sentiment analysis: A survey," in *Proceedings of the 2019 International Conference on Artificial Intelligence, Robotics and Control*, 2019, pp. 39–43.
- [8] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, pp. 3111–3119, 2013.