Towards Procedural Generation of Constructed Languages for Games

Aaron Cai, Chris Martens

Northeastern University
Boston, MA 02115, USA

Abstract

Fictional languages are an aspect present in many hand-authored narratives, but underexplored in procedurally generated narratives. This paper presents an initial exploration of procedurally generated constructed languages (conlangs), following principles of language construction that relate phonology, morphology, and syntax. We present this generator as a tool for game developers. Our approach allows procedurally generated worlds and narratives to draw upon a generative space of languages that follow plausible linguistic principles.

Keywords

procedural generation, constructed languages, game worlds, worldbuilding

1. Introduction

Research and game development has come a long way for procedural generation of detailed game worlds that support rich narrative experiences, such as Dwarf Fortress [1], with its interconnected systems of geology, ecology, and history that inform how civilizations develop and interact [2]. These worlds often include simulations of human-like societies that develop social identity, culture, and tradition. However, the spoken and written languages of these cultures remain an under-explored feature for procedural world generation. In Dwarf Fortress, for example, the set of four languages (one for each race) is completely fixed and only varies in terms of alphabet and lexemes (i.e., each of the languages is a relexification [3] of the others).

We propose that constructed languages, or conlangs for short, offer a window into how plausible languages might be generated for fictional societies in games. In this paper, we describe a work-in-progress generator based on documented conlang methods that can create a new and unique conlang each time generation is invoked. We present the generator in its current form as a tool for game developers.

One specific consideration for language generation in the context of game-worlds is the aesthetics of the language. For example, people who have a lot of exposure to languages with shared aesthetic features will be able to pick up the sound patterns and tendencies of those languages and be able to distinguish that language from a lineup without necessarily actually understanding the spoken words. Those who understand more than one language will know that sentences or words may be structured differently, or that certain information is required grammatically in one language but entirely optional in others. These factors all contribute to the aesthetic of the language, and can make a language feel out of place in certain contexts or appropriate to others. We would like for game developers that use this tool to be able to supply parameters so that it creates a language with a fitting aesthetic for the context it will be used in.

Another goal for this project is to mimic patterns in natural human languages so that it can generate languages that players perceive as believable. Therefore, the creation of the generator is informed by research in the field of linguistics and is thus separated into sections that deal with phonology, morphology, and syntax. Where possible, the models in the generator were informed by empirical data. This mimicry has the added benefit of making the tool easier to understand in terms of intuitions about human languages, as well as allowing the generator to make decisions based on data from real human languages.

The vision for this tool is so that it can be used in games in such a way that players will encounter a different conlang each time they play. One potential application is language-decoding puzzle gameplay (such as that found in Heaven's Vault [4]) based on the conlang, in a way that is robust to replay and to having answers posted online. However, there are also more general opportunities to have the language of a generated civilization reflect aspects of its culture, to include the writing in in-world artifacts (such as signage and books), and to see it written or spoken by NPCs. In the long term, we imagine this tool supporting games that include procedurally generated worlds and narratives enriched by unique languages.

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

cai.a@northeastern.edu (A. Cai); c.martens@northeastern.edu (C. Martens)

CEUR Workshop Proceedings (CEUR-WS.org)

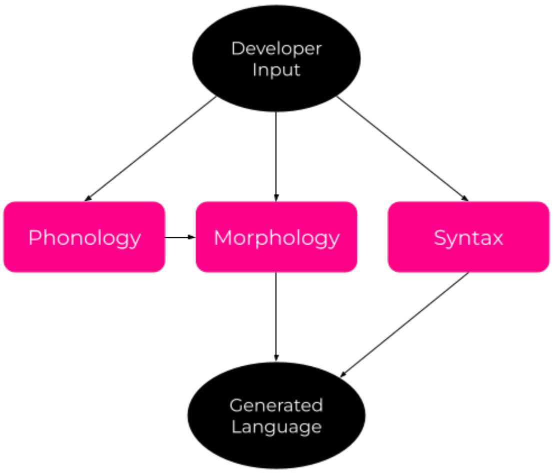


Figure 1: Dataflow architecture of GenLang.

2. Related Work

The project of procedurally generating fictional languages for game worlds appears to be in its infancy, but we have identified two previous publications on the topic. First, Mark Johnson's development on *Ultima Ratio Regum* (URR), includes the generation of dialects and naming conventions for a procedurally-generated fictional society [5]. *URR* constructs and selects syllables at random from a process constrained by arbitrary hand-authored decisions. By contrast, our work demonstrates a linguistics-informed process of sound selection, phonotactics, and concern for plausibility with respect to real-world human language development. Second, James Ryan describes a system for simulating the process of human societies naturally developing and evolving language [6]. By contrast, the system we describe operationalizes the practice of designing *constructed languages*, which exist at a level of abstraction above direct simulation of human language development. In fact, Ryan mentions the possibility of generating conlangs as a ripe opportunity for future work.

3. Overview

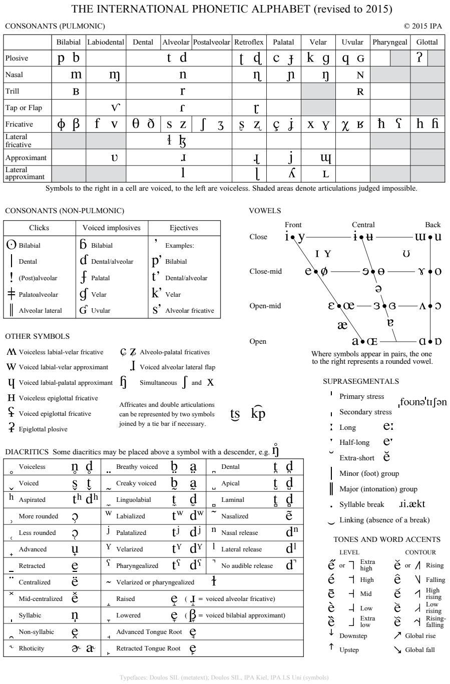
In the course of this paper, there will be discussion about potential game developers who could use this tool to assist in the creation of games, as well as players who could play those created games. For the purpose of clarity, game developers will be henceforth referred to as developers, and game players will be referred to as players. It is important to note that developers will not refer to us, the authors and creators of the generator. We will avoid the term "users", even though it may be natural to think of developers as such.

The creation of a constructed language generator necessarily includes digitally modeling the structure of language at multiple levels of abstraction, such as phonology, vocabulary, grammar, and writing systems. A small change in one of these models can influence others in global ways. As such, our system of models needs to communicate with each other effectively. This informs our decision to design the architecture of the software as a hierarchical one, where code modules handle cascading levels of abstraction. An overarching script handles the passing of information between the different modules as well as the developer (see Figure 1). The higher-level modules (macro-modules) include phonology, morphology, and syntax. These terms may be unfamiliar for those without a linguistics background, and they will be explained in greater detail in their respective sections. The general flow of execution starts from gathering developer input, and providing that information to the phonology macro-module. Developer input is stored in an object that comprises of arrays of strings, booleans, integers, and enums. The strings contain information regarding settings pertaining to specific sounds, and the other data types handle more easily quantified settings. The outputs of the phonology macro-module as well as already gathered developer input will then be provided to the morphology macro-module. The syntax macro-module also uses developer input but does not need the output of the other macro-modules. The outputs of the morphology and syntax modules are then used to generate output that the developer and players will see. The exact structure of the data passed between these modules will be discussed in further detail later. Each macro-module is subdivided into lower-level modules (micro-modules), the specifics of which will be discussed in their respective sections.

Our research that informs the generator includes recent, empirical linguistics studies, avoiding older or more qualitative studies that have been criticized for bias (e.g. Anglocentrism). That is not to say that recent empirical studies are not without bias, but in lieu of directly consulting linguistic experts, we take them to represent the most up-to-date knowledge about human languages, and summarize the aspects of these findings relevant to our project in each of the subsequent sections.

4. Phonology

The phonology of a language is all the sounds, or phonemes, that speakers of the language use when speaking that language. Phonology can differ greatly between languages and is one of the first things to be noticed by someone who does not understand a language being spoken. Therefore, take phonology as the biggest



THE INTERNATIONAL PHONETIC ALPHABET (revised to 2015)

CONSONANTS (PULMONIC) © 2015 IPA

	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Retroflex	Palatal	Velar	Uvular	Pharyngeal	Glottal
Plosive	p b			t d		ʈ ɖ	c ɟ	k ɡ	q ɢ		ʔ
Nasal	m	ɱ		n		ɳ	ɲ	ŋ	ɴ		
Trill	ʙ			r					ʀ		
Tap or Flap		ⱱ		ɾ		ɽ					
Fricative	ɸ β	f v	θ ð	s z	ʃ ʒ	ʂ ʐ	ç ʝ	x ɣ	χ ʁ	ħ ʕ	h ɦ
Lateral fricative				ɬ ɮ							
Approximant		ʋ		ɹ		ɻ	j	ɰ			
Lateral approximant				l		ɭ	ʎ	ʟ			

Symbols to the right in a cell are voiced, to the left are voiceless. Shaded areas denote articulations judged impossible.

CONSONANTS (NON-PULMONIC)

Clicks	Voiced implosives	Ejectives
ʘ Bilabial	ɓ Bilabial	ʼ Examples:
ǀ Dental	ɗ Dental/alveolar	pʼ Bilabial
ǃ (Post)alveolar	ʄ Palatal	tʼ Dental/alveolar
ǂ Palatoalveolar	ɠ Velar	kʼ Velar
ǁ Alveolar lateral	ʛ Uvular	sʼ Alveolar fricative

VOWELS

Where symbols appear in pairs, the one to the right represents a rounded vowel.

OTHER SYMBOLS

ʍ Voiceless labial-velar fricative
w Voiced labial-velar approximant
ɥ Voiced labial-palatal approximant
ʜ Voiceless epiglottal fricative
ʢ Voiced epiglottal fricative
ʡ Epiglottal plosive

ɕ ʑ Alveolo-palatal fricatives
ɺ Voiced alveolar lateral flap
ɧ Simultaneous ʃ and x

Affricates and double articulations can be represented by two symbols joined by a tie bar if necessary. t͡s k͡p

SUPRASEGMENTALS

ˈ Primary stress ˌfoʊnəˈtɪʃən
ˌ Secondary stress
ː Long eː
ˑ Half-long eˑ
˘ Extra-short ĕ
| Minor (foot) group
‖ Major (intonation) group
. Syllable break ɹi.ækt
‿ Linking (absence of a break)

DIACRITICS Some diacritics may be placed above a symbol with a descender, e.g. ŋ̊

◌̥ Voiceless	n̥ d̥	◌̤ Breathy voiced	b̤ a̤	◌̪ Dental	t̪ d̪
◌̬ Voiced	s̬ t̬	◌̰ Creaky voiced	b̰ a̰	◌̺ Apical	t̺ d̺
ʰ Aspirated	tʰ dʰ	◌̼ Linguolabial	t̼ d̼	◌̻ Laminal	t̻ d̻
◌̹ More rounded	ɔ̹	ʷ Labialized	tʷ dʷ	◌̃ Nasalized	ẽ
◌̜ Less rounded	ɔ̜	ʲ Palatalized	tʲ dʲ	ⁿ Nasal release	dⁿ
◌̟ Advanced	u̟	ˠ Velarized	tˠ dˠ	ˡ Lateral release	dˡ
◌̠ Retracted	e̠	ˤ Pharyngealized	tˤ dˤ	◌̚ No audible release	d̚
◌̈ Centralized	ë	~ Velarized or pharyngealized	ɫ		
◌̽ Mid-centralized	e̽	◌̝ Raised	e̝ (ɹ̝ = voiced alveolar fricative)		
◌̩ Syllabic	n̩	◌̞ Lowered	e̞ (β̞ = voiced bilabial approximant)		
◌̯ Non-syllabic	e̯	◌̘ Advanced Tongue Root	e̘		
˞ Rhoticity	ɚ a˞	◌̙ Retracted Tongue Root	e̙		

TONES AND WORD ACCENTS

LEVEL		CONTOUR	
e̋ or ˥	Extra high	ě or ˩˥	Rising
é ˦	High	ê ˥˩	Falling
ē ˧	Mid	e᷄ ˦˥	High rising
è ˨	Low	e᷅ ˩˨	Low rising
ȅ ˩	Extra low	e᷈ ˧˦˧	Rising-falling
↓	Downstep	↗	Global rise
↑	Upstep	↘	Global fall

Typefaces: Doulos SIL (metatext); Doulos SIL, IPA Kiel, IPA LS Uni (symbols)

Figure 2: IPA Chart showing how to use IPA notation, http://www.internationalphoneticassociation.org/content/ipa-chart, available under a Creative Commons Attribution-Sharealike 3.0 Unported License. Copyright © 2015 International Phonetic Association

contributing factor to the aesthetic of a game-world language when the player first encounters it. In settings with verbalized NPC dialog, it would likely be the most salient aspect of the language (prior to the player learning the language's structure and semantics). One aspect of phonology that is easily modeled digitally is the size of the sound inventory. This describes the amount of different sounds speakers of the language make while speaking that language. Languages can differ in this way by incredible magnitudes. Some languages have as few as 6 consonants, for example, while others have more than 100 [7]. As for which sounds are available for selection, most linguists use the International Phonetic Alphabet (IPA) [8] (see Figure 2). All sounds that have been observed by linguists to occur in any human language can be notated using IPA [9]. Looking through all the IPA symbols gives us a list of sounds to select from. It is of note that the IPA does not have a symbol for every possible sound a human mouth can make: there are a few sounds that are technically possible but have not been observed in any human language.

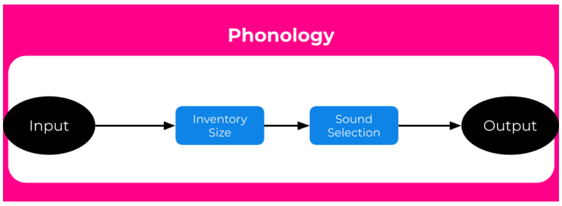


Figure 3: Dataflow between submodules of the phonology module.

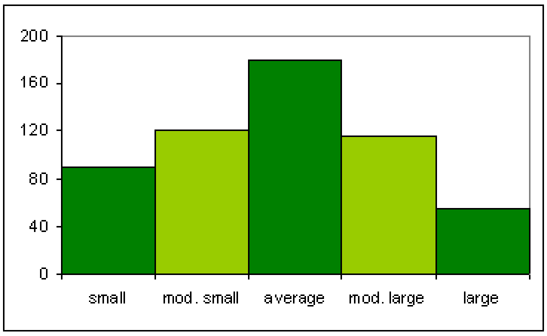


Figure 4: Histogram of languages according to categories of consonant inventory size. The small category indicates an inventory of 6-14 consonants, a moderately small inventory contains between 15-18, an average one contains 19-25, a moderately large one contains 26-33, and a large one contains between 34-122.

4.1. Implementation

The overall structure of this macro-module is a linear path from one micro-module to the next (see Figure 3). We start off with sound inventory size, then move onto sound selection. The output is a phonology object which stores all the sounds within the generated language.

If the developer does not specify a size for the phoneme inventory, the tool will choose one based on a distribution derived from a database [7] which surveyed real languages all around the world (see Figure 4). The distribution appears to be a normal distribution at first glance, but the categories are not equally divided. The "large" category actually includes a much larger range of numbers than the other categories. One could attempt to regress a skewed probability distribution, but picking a category with a distribution matching the data and then picking a number in that category in an uniform distribution approximates it sufficiently.

Phonemes are modeled in the generator using objects that store information such as a long, descriptive name for ease of handling within the code, the IPA notation, and information about how the sound is created using the mouth. The latter information is not immediately important but will become relevant in other parts of the generator. These objects can be of either the consonant type or the vowel type, but both will eventually make up a phoneme object. The reason why this additional level of abstraction is necessary is due to the fact that some languages treat multiple consonants or vowels as a singular phoneme. For example, the /tʃ/ sound in English is treated as a single phoneme by English despite typically being spelled with two alphabetic consonants ("ch") and being composed of two IPA sounds. The generator will also exclude phonemes that have never been observed in any human language. This set is the same every time the generator is run, so the information is stored in a JSON document that the script reads upon execution.

In terms of actually selecting the sounds, some sounds are more common than others. When the selection of sounds is not specified by the developer, the generator picks sound in a distribution informed by a database of phonological data of real human languages [10]. A disclaimer from the database website that is well-included here is that certain languages have multiple entries from disagreeing sources. This would mean that languages under-studied would not influence the data as much as over-studied languages. However, we decided not to remove these multiple entries, assuming that such discrepancies will not be noticed by players. One could weigh the influence of a language by using population data, but this could bias the dataset in a Eurocentric direction, since the effects of colonialism pervade the landscape of real-world languages. One could also reduce the weight of individual entries if a language has multiple, but this would ultimately be a subjective decision, since boundaries between related languages can be hard to determine. Linguists are still debating over what counts as a language, a dialect, or an accent [11]. If linguistics research provides a better database to inform the generator in the future, it would be worthwhile to switch.

Once all the phonemes are selected are selected, they are stored in a phonology object, which is in essence an array of phoneme objects with some wrapper functions. The phonology object is returned and can then be used as the input for the morphology.

5. Morphology

Morphology is the study of the construction of words. Words are not easy to classify as they can differ in definition or purpose between different languages [12]. For example, large numbers in English are expressed in what is defined as multiple words, but those same numbers in German are expressed in what is defined as one word. For this reason, the tool does not deal with words but with syllables and morphemes, which are more concretely defined in linguistics. Morphology is an aspect of language that becomes more relevant to game-world languages when the player is intended to learn to speak or write the language.

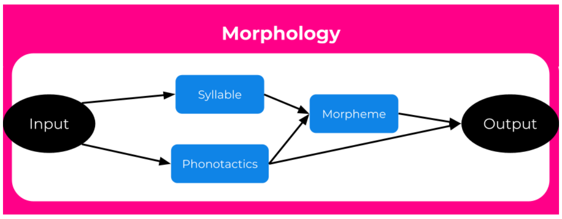


Figure 5: Dataflow between submodules of the Morphology module.

In our implementation, the morphology macro-module is less linear than the phonology module. It consists of three micro-modules: morpheme, syllable, and phonotactics. The primary function lies in the morpheme micro-module, which uses the syllable and phonotactics micro-modules in its execution. The output is a morphology object which contains the outputs of both the phonotactics and morpheme micro-modules (see Figure 5).

5.1. Syllables

Syllables may seem like a concept that is the same across all languages, but different languages define syllables in different ways. Certain languages may define a series of vowels as a diphthong, which would count for only one syllable, while others count that same series of vowels as separate syllables (e.g., /aɪ/ as in "my" in English compared to /aɪ/ as in "愛" in Japanese).

In addition, consonants at the beginning of a syllable may have different rules than consonants at the end of a syllable. Different languages can have rules on how vowels appear in syllables as well. English can have three vowels as a triphthong in a single syllable as in the word "flour," but other languages such as Japanese allows only a singular vowel sound in a syllable. In addition, some languages, such as Mandarin, use tones. Tonal languages differentiate syllables with the pitch or change in pitch of articulation.

Even with the overall structure of the syllable decided, there may be additional rules on which combinations of sounds are allowed together. For example, even though English allows for a consonant cluster of two at the beginning of a syllable, an "s" sound followed by an "r" sound is not allowed. In addition, in many languages (including English), consonants can come in clusters that still contribute to only a single syllable, but the maximum number of consonants in a single cluster differs by language.

5.1.1. Implementation.

For the sake of simplicity, the generator regards syllables as a sequence of vowels with potentially a sequence of consonants on either side. Due to the way that languages differ greatly in syllable construction rules, the generator encourages developers to specify how many consonants can be at the beginning and end of each syllable, as well as how many vowels can appear in the middle. They can also specify whether tones are differentiating within the language, and how many and which tones are used.

Within a single language, certain syllable structures may be more common than others. For example, English speech has many more syllables starting with one or two consonants than with three. Due to lack of data on this matter, the generator uses a Zipfian distribution, which is a distribution observed in many natural distributions, such as frequency of a word within bodies of text, or population of cities. In a Zipfian distribution, the frequency of occurrence of an element is inversely proportional to its rank in a frequency table. The generator stores these syllables as an object which primarily consists of a sequence of phonemes. However, unlike phonemes, not all possible syllables within the language are stored as the number of possible syllables may be orders of magnitude higher than that of possible phonemes, depending on the rules of syllable construction.

5.2. Phonotactics

Phonotactics refers to such rules as which sounds are allowed together, as well as how sounds can change in certain contexts [13]. For example, the vowel in the second syllable of “modal” changes when an additional part is added onto the end as in “modality.” There is an incredibly large array of such rules for change across real human languages [14]. We did not find a database documenting the frequency of such rules across languages, so we were not able to weigh the probability of phonotactic rules being picked by the generator by their frequency in real human languages.

5.2.1. Implementation

The phonotactics micro-module includes a few common phonotactic rules that we deemed worth emulating, e.g. the tendency for unvoiced consonants to become voiced when they appear between multiple occurrences of the same vowel.

If the developer does not specify phonotactic rules, the generator will pick several, but not all, at random to be enabled. The generator will first pick a number between a quarter to a half of the total number of rules available to pick from. It will then pick that many using a uniform distribution. There may be correlation between the nature of the sound inventory and the phonotactic rules, but that may be unnecessary for believability and is beyond the expertise of the authors. The phonotactics rules are stored as a phonotactics object consisting of multiple rules stored as strings. When a syllable is constructed, the syllable micro-module checks to make sure that every rule is followed.

There may be a more computationally efficient method of storing these rules than as strings, but they are easy to work with in development. An optimization pass in the future may change the way these are stored

5.3. Morphemes

Morphemes are defined by linguists as atomic units of meaning [15]. The “s” sound that English speakers attach onto the end of words to indicate plurality is an example of a morpheme. Evidently, morphemes can consist of multiple syllables, a singular syllable, or even a consonant or vowel that changes another morpheme’s syllable. Because they are so flexible, morphemes are stored as an object that consists of either phonemes or syllables, with a property indicating how they are to be fitted with other morphemes. This can specify that they are free, which means they can stand alone in speech and are easy to deal with. However, bound morphemes cannot stand alone and thus also store rules on how they are to be attached to other morphemes.

5.3.1. Implementation.

With the tools of syllable construction and phonotactic rules, we can begin to construct morphemes. In the morpheme micro-module, the generator can finally assign meaning to sounds. Morphemes are stored as objects with strings indicating its pronunciation and meaning, as well as an enum for its grammatical type. After a basic set of morphemes are created, they are stored in an morphology object, which also stores the phonotactics object from the previous submodule. The morphology object can then be passed on to the final output

6. Syntax

Syntax is the subdiscipline of linguistics that deals with the construction of sentences from morphemes [16]. Syntax can differ between languages to a degree that may surprise those who speak one language. For example, English adjectives typically appear before the noun it is describing, but in Romance languages, adjectives typically occur after the described noun [17]. One easy aspect of syntax to model digitally is word order, i.e. the order things appear in sentences. English sentences typically start with a subject, followed by the verb, and then the object. This order can be abbreviated to SVO (Subject, Verb, Object). However, this is not the only possibility.

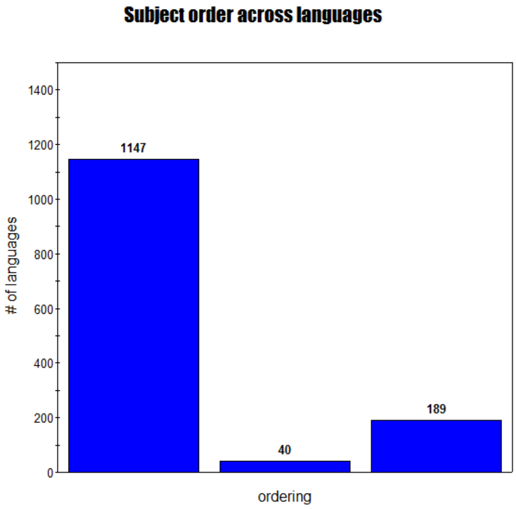


Figure 6: Subject Object order across languages

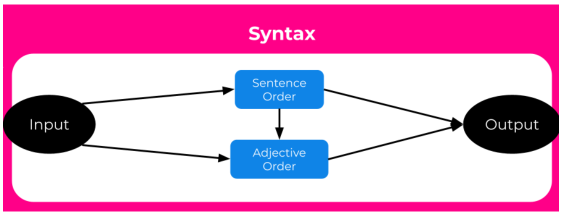


Figure 7: Dataflow between submodules of the Syntax module.

There are six ways to arrange an ordering of three distinct objects. Therefore, there are six possible ways to order the subject, verb, and object in a sentence. While all permutations have been observed in real human languages, some orders are far more common than others. Notably, it seems that vast majority of human languages have the subject occur before the object [18] (see Figure 6).

6.1. Implementation

The syntax macro-module creates the rules for the construction of sentences (see Figure 7). While it may seem necessary for the macro-module to use the morphology macro-module, that is not actually the case. The syntax module is solely responsible for the rules with which morphemes or morphemes clusters are ordered in sentences, regardless of what the morphemes or morpheme clusters are comprised of. Syntax is something that can greatly increase the feeling of foreignness for players if they are learning to understand the language.

For sentence word order, we include both the common options (SVO, SOV, and VSO), and the uncommon (OVS, OSV, VOS). The generator will first decide which of these categories to pick from, weighted heavily towards the common category. It will then pick uniformly at random within the category. For developers that choose to specify a sentence order, the generator produces some text offering guidance to not use the ones with the object before the subject, and suggests that using a different sentence-order from the player's native language might make it harder to interpret or relate to (leaving the judgment of whether or not this property is desired to the developer).

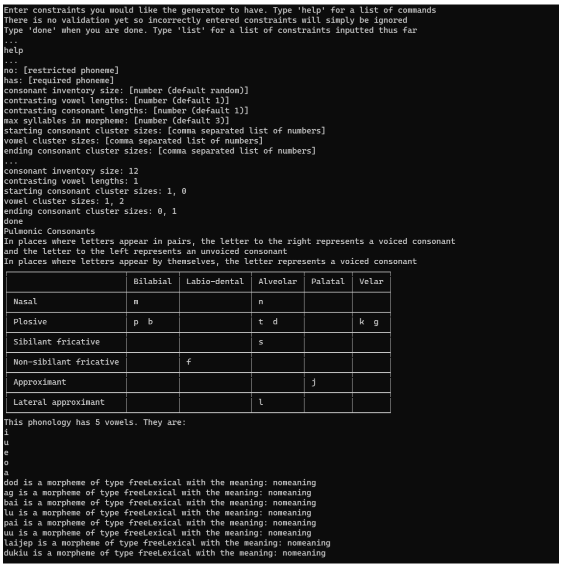
After the sentence ordering is decided, the generator chooses the ordering of modifiers (e.g. adjectives, adverbs) relative to the head, or the part of the sentence being modified. (We distinguish the head from nouns because linguists include other parts of speech such as verbs as a possibility for the head.) In the initial stages of this project, this ordering was affected by the sentence word order as informed by older studies that have since been challenged. Informed by more recent literature [19], the generator will pick modifier-head order at random without consideration for sentence word order. These decisions will be stored in a syntax object that is outputted and can then be used in conjunction with a morphology object to create sentences. While there are many other ways languages differ in syntax, these two aspects provide a starting point that we intend to develop further. Some potential additions include the ordering of numerals and nouns [20], whether nominal and verbal conjunction are different [21], and different manifestations of associative plurals [22].

These rules are stored in a syntax object which composes of an enum for sentence word order and a boolean for the ordering of modifiers. The syntax object can then be passed on to the final output.

7. Implementation Status

We are in the progress of implementing the design described in this paper as the GenLang tool available on GitHub at https://github.com/AkaiGameDev/GenLang. Currently, we have implemented the Phonology module and the Morphology module except for Phonotactics. Phonotactics and Syntax remain to me implemented.

Figure 8 shows the current command-line interface and corresponding output. Each prompt ending in ":" represents an opportunity for developer input. The system then prints out the generated phonology including a 2-dimensional consonant chart, sample morphemes generated and associated meanings. See Appendix A for a second input-output example in text form.



```
Enter constraints you would like the generator to have. Type 'help' for a list of commands
There is no validation yet so incorrectly entered constraints will simply be ignored
Type 'done' when you are done. Type 'list' for a list of constraints inputted thus far
...
help
...
no: [restricted phoneme]
has: [required phoneme]
consonant inventory size: [number (default random)]
contrasting vowel lengths: [number (default 1)]
contrasting consonant lengths: [number (default 1)]
max syllables in morpheme: [number (default 3)]
starting consonant cluster sizes: [comma separated list of numbers]
vowel cluster sizes: [comma separated list of numbers]
ending consonant cluster sizes: [comma separated list of numbers]
...
consonant inventory size: 12
contrasting vowel lengths: 1
starting consonant cluster sizes: 1, 0
vowel cluster sizes: 1, 2
ending consonant cluster sizes: 0, 1
done
Pulmonic Consonants
In places where letters appear in pairs, the letter to the right represents a voiced consonant
and the letter to the left represents an unvoiced consonant
In places where letters appear by themselves, the letter represents a voiced consonant
```

	Bilabial	Labio-dental	Alveolar	Palatal	Velar
Nasal	m		n		
Plosive	p b		t d		k g
Sibilant fricative			s		
Non-sibilant fricative		f			
Approximant				j	
Lateral approximant			l		

```
This phonology has 5 vowels. They are:
i
u
e
o
a
dod is a morpheme of type freeLexical with the meaning: nomeaning
ag is a morpheme of type freeLexical with the meaning: nomeaning
bai is a morpheme of type freeLexical with the meaning: nomeaning
lu is a morpheme of type freeLexical with the meaning: nomeaning
pai is a morpheme of type freeLexical with the meaning: nomeaning
uu is a morpheme of type freeLexical with the meaning: nomeaning
laijep is a morpheme of type freeLexical with the meaning: nomeaning
dukiu is a morpheme of type freeLexical with the meaning: nomeaning
```

Figure 8: A screenshot of the current GenLang command-line user interface and structured output.

8. Discussion

Our main contribution is a preliminary algorithm, designed as a set of modules, for procedurally generating a constructed language for use in a game world. However, this project is very much a work in progress, and also entails some inherent limitations.

First, while we have done some preliminary exploration of written language generation, it is not incorporated into this generator and brings its own set of challenges. A written language generator would have to generate 2D artifacts with constraints that are difficult to proceduralize. Written languages typically have glyphs that are easy to write and distinguish from each other. It may be difficult to algorithmically evaluate how well generated artifacts satisfy those constraints. A written language generation could be integrated with our generator to allow for a more sophisticated generated language, as well as supporting gameplay with more emphasis on visual communication than audio (e.g. text games).

Second, there are important aspects of syntax and semantics that we have not yet implemented, or in some cases designed good solutions for. For example, the problem of generating a lexicon (set of meanings to assign to morphemes) is currently the responsibility of the developer, and currently lexicon meanings will be randomly assigned to morphemes. But this approach does not take into account the possible relationships between morphemes, including syntactic concerns like stemming, inflection, conjugation, tense, aspect, and mood. These choices could impact other aspects of syntax; for example, if a verb can be conjugated to encode the subject-pronoun (as in Spanish), the pronoun subject can be dropped ("pro drop"). Additionally, we do not treat aspects of vocabulary that might be affected by the cultural context of the world for which the language is being developed. Nor do we treat any aspect of semantics, e.g. a process for constructing character utterances appropriate to specific game-world contexts, although this task is one over which we assume a developer would usually want more manual control.

Finally, GenLang is a project by humans with inherent biases, so it cannot possibly avoid linguistic bias nor fully represent the diversity of human languages. Therefore, this generator comes with the disclaimer that the creators are not experts in linguistics, and (across all authors) are most familiar with English, Mandarin, Japanese, and Spanish (our "source languages"). There are going to be unaccounted-for aspects of real-world languages and linguistic topics that the authors are unfamiliar with. Fortunately, our source languages differ from each other in numerous ways and thus provide insight into how languages can differ. However, it is likely that the differences across our source languages are emphasized in the design of the generator and the similarities are overlooked. If others with an entirely different set of source languages and linguistic or computational knowledge were to tackle the challenge of algorithmically modeling languages, they may focus more on other aspects of languages, resulting in a different algorithm. In addition, the authors acknowledge that linguists' understanding of languages is still changing, and that even models made by practicing linguists on well-studied real human languages are not without flaws. As this project replicates such models, it follows that the outputs of this generator may include such flaws as well.

9. Conclusion

We have presented a proof-of-concept generator for phonology, morphology, and syntax in a constructed language, drawing from empirical linguistics research to represent a cross-section of the possibility space of languages spoken on Earth. Taking into account the intended use case of generating languages within fictional game worlds, we have presented our design decisions for this tool. We demonstrate this design with a working software implementation.[1] As a result, this project makes significant inroads towards the longer-term ambition of generating complete, novel constructed languages that incorporate developer input.

[1] Available at https://github.com/AkaiGameDev/GenLang.

References

[1] B. . Games, Dwarf Fortress, PC, 2006.

[2] T. Betts, Procedural content generation, Handbook of Digital Games (2014) 62–91.

[3] P. H. Matthews, The concise Oxford dictionary of linguistics, Oxford Quick Reference, 2014.

[4] Inkle, Heaven's Vault, PC, PlayStation 4, Nintendo Switch, 2019.

[5] M. R. Johnson, Procedural generation of linguistics, dialects, naming conventions and spoken sentences, in: Proceedings of the FDG workshop on Procedural Content Generation, 2016.

[6] J. Ryan, Diegetically grounded evolution of gameworld languages, Proc. Procedural Content Generation (2016).

[7] I. Maddieson, Consonant inventories (v2020.3), in: M. S. Dryer, M. Haspelmath (Eds.), The World Atlas of Language Structures Online, Zenodo, 2013. URL: https://doi.org/10.5281/zenodo.7385533. doi:10.5281/zenodo.7385533.

[8] P. T. Daniels, W. Bright, The world's writing systems, New York: Oxford University Press, 1996.

[9] I. P. Association, Handbook of the International Phonetic Association: A guide to the use of the International Phonetic Alphabet, Cambridge: Cambridge University Press, 1999.

[10] S. Moran, D. McCloy (Eds.), PHOIBLE 2.0, Max Planck Institute for the Science of Human History, Jena, 2019. URL: https://phoible.org/.

[11] S. R. Anderson, How many languages are there in the world?, Linguistic Society of America (2010).

[12] M. van Oostendorp, Words and sentences, 2023.

[13] B. Hayes, Introductory Phonology, Wiley-Blackwell, 2008, p. 64.

[14] M. R. Freeman, H. K. Blumenfeld, V. Marian, Phonotactic constraints are activated across languages in bilinguals, Frontiers in Psychology 7 (2016). doi:https://doi.org/10.3389/fpsyg.2016.00702.

[15] M. van Oostendorp, Morphology, 2023.

[16] M. van Oostendorp, Syntax and word order (a), 2023.

[17] M. van Oostendorp, Syntax and word order (b), 2023.

[18] M. S. Dryer, Order of subject, object and verb (v2020.3), in: M. S. Dryer, M. Haspelmath (Eds.), The World Atlas of Language Structures Online, Zenodo, 2013. URL: https://doi.org/10.5281/zenodo.7385533. doi:10.5281/zenodo.7385533.

[19] M. S. Dryer, Relationship between the order of object and verb and the order of adjective and noun (v2020.3), in: M. S. Dryer, M. Haspelmath (Eds.), The World Atlas of Language Structures Online, Zenodo, 2013. URL: https://doi.org/10.5281/zenodo.7385533. doi:10.5281/zenodo.7385533.

[20] M. S. Dryer, Order of numeral and noun (v2020.3), in: M. S. Dryer, M. Haspelmath (Eds.), The World Atlas of Language Structures Online, Zenodo, 2013. URL: https://doi.org/10.5281/zenodo.7385533. doi:10.5281/zenodo.7385533.

[21] M. Haspelmath, Nominal and verbal conjunction (v2020.3), in: M. S. Dryer, M. Haspelmath (Eds.), The World Atlas of Language Structures Online, Zenodo, 2013. URL: https://doi.org/10.5281/zenodo.7385533. doi:10.5281/zenodo.7385533.

[22] M. Daniel, E. Moravcsik, The associative plural (v2020.3), in: M. S. Dryer, M. Haspelmath (Eds.), The World Atlas of Language Structures Online, Zenodo, 2013. URL: https://doi.org/10.5281/zenodo.7385533. doi:10.5281/zenodo.7385533.

A. Worked Example

Example developer configuration:

```
 - Consonant inventory size: Average
- Vowel inventory size: 5
- Include: ['s', 'k']
- Exclude: []
- Beginning consonants: [1]
- Ending consonants: [0]
- Consecutive vowels: [1]
- Tonality: None
- Phonotactics: []
- Sentence order: ['SVO']
- Modifier order: ['HM']
```

Possible Output:

```
selected 22 consonants and 5 vowels.
Consonants:
IPA character: l
Descriptive name:
    VoicedAlveolarLateralApproximant
Place of Artic.: ['alveolar']
Manner:     ['lateralapproximant']


IPA character: m
Descriptive name: VoicedBilabialNasal
Place of Artic.: ['bilabial']
Manner:     ['nasal']


IPA character: s
Descriptive name: VoicelessAlveolarSibilant
Place of Artic.: ['alveolar']
Manner:     ['sibilant']
```

```
IPA character: b
Descriptive name: VoicedBilabialPlosive
Place of Artic.: ['bilabial']
Manner:     ['plosive']


IPA character: n
Descriptive name: VoicedAlveolarNasal
Place of Artic.: ['alveolar']
Manner:     ['nasal']


IPA character: k
Descriptive name: VoicelessVelarPlosive
Place of Artic.: ['velar']
Manner:     ['plosive']


IPA character: t
Descriptive name: VoicelessAlveolarPlosive
Place of Artic.: ['alveolar']
Manner:     ['plosive']


IPA character: j
Descriptive name: VoicedPalatalApproximant
Place of Artic.: ['palatal']
Manner:     ['approximant']


IPA character: d
Descriptive name: VoicedAlveolarPlosive
Place of Artic.: ['alveolar']
Manner:     ['plosive']


IPA character: g
Descriptive name: VoicedVelarPlosive
Place of Artic.: ['velar']
Manner:     ['plosive']


IPA character: f
Descriptive name:
    VoicelessLabiodentalFricative
Place of Artic.: ['labiodental']
Manner:     ['fricative']


IPA character: p
Descriptive name: VoicelessBilabialPlosive
Place of Artic.: ['bilabial']
Manner:     ['plosive']


IPA character: g!
Descriptive name: VoicedAlveolarClick
Place of Artic.: ['alveolar', 'velar']
Manner:     ['click']
```

```
IPA character: ɠ
Descriptive name: VoicedVelarImplosive
Place of Artic.: ['velar']
Manner:     ['implosive']


IPA character: ɻ
Descriptive name: VoicedRetroflexApproximant
Place of Artic.: ['retroflex']
Manner:     ['approximant']


IPA character: ʈ'
Descriptive name: RetroflexEjectiveStop
Place of Artic.: ['retroflex']
Manner:     ['plosive', 'ejective']


IPA character: ʕ
Descriptive name: VoicedPharyngealFricative
Place of Artic.: ['pharyngeal']
Manner:     ['fricative']


IPA character: ʂ
Descriptive name: VoicelessRetroflexSibilant
Place of Artic.: ['retroflex']
Manner:     ['sibilant']


IPA character: ç'
Descriptive name: PalatalEjectiveFricative
Place of Artic.: ['palatal']
Manner:     ['fricative', 'ejective']


IPA character: q
Descriptive name: VoicelessUvularPlosive
Place of Artic.: ['uvular']
Manner:     ['plosive']


IPA character: ʐ
Descriptive name: VoicedRetroflexSibilant
Place of Artic.: ['retroflex']
Manner:     ['sibilant']


IPA character: r̯
Descriptive name: VoicelessAlveolarTrill
Place of Artic.: ['alveolar']
Manner:     ['trill']


Vowels:
i
u
e
o
```

```
a

Generated 10 words:
r̯elime - meaning observe
ɠa - meaning man
ge - meaning woman
ju - meaning child
ɠosa - meaning bread
togi - meaning cook
kitijo - meaning good
ʐogibaso - meaning bad
ɕ’o - meaning eat
qa - meaning - meaning many


Generated 3 sentences:

English translation:
The man bakes good bread
Generated language:
ɠa togi ɠosa kitijo

English translation:
The child eats bad bread
Generated language:
ju ɕ’o ʐogibaso ɠosa

English translation:
The woman watches the child cook
Generated language:
ge r̯elime ju togi
```