yod: A library for language generation

Daniel James

April 2017

Chapter 1

Introduction

yod is a library, written in `C#`, designed to aid in the construction and generation of artificial languages. Artificial languages (also known as constructed languages or 'conlangs') are probably most publicly well known through the popularisation of 'fictional languages' - that is, languages designed for use in works of fiction, usually for adding depth to a fictional world. For example, J. R. R. Tolkien devised the languages of *Sindarin* and *Quenya* for use in his *Lord of the Rings* series. Other popular fictional languages that have reached mainstream awareness include *Star Trek*'s *Klingon* and *Game of Thrones*' *Dothraki*.

The creation of these languages has become an indispensable tool for writers to add depth and character to their worlds. However, constructed languages do not exist solely for creative uses. The language *Esperanto* was developed by its creator L. L. Zamenhof with the goal of being a global language that was easy for people to learn[1]. Now it is reported that as many as 63,000 people worldwide can speak Esperanto to some degree[2]. The existence of Esperanto and other so-called "international auxiliary languages" (languages designed to simplify complication between people of different countries and cultures) shows that the usefulness of constructed languages extends into real-world applications as well as artistic uses.

yod was developed with a focus on artistic languages, and so the goal was to generate a large variety of languages which could suit many different worlds and fictional civilisations. However, due to the hierarchy of rules *yod* uses to build languages, the languages are regular, in contrast to most natural languages, which means generated languages with certain features could be treated as auxiliary languages too.

As a library, *yod* provides interfaces to easily generate any part of a language, including phonology, orthography and grammar and sub-parts thereof. This works by having each class which represents a randomisable constituent exposing a `Generate()` method which creates a new instance of that class from scratch with procedurally generated properties. If necessary, this sometimes also means recursively generating that objects constituent members. Writing these generator methods in a way that takes into account certain restrictions or statistics allows for much more realistic values for the object.

Merely generating a new language every time the library is used, however, would not allow for enough control over the language generation process for users who are more experienced with the concept of ’conlangs’ and the construction thereof. It also does not allow for iterative language-building, wherein the user keeps features of a generated language that they like, but re-generates the rest of the features. This would allow a user to quickly and easily refine their language until it suited their use case.

To facilitate this, *yod* uses a data-driven approach which allows almost every class that can be randomly generated to also be serialised to and deserialised from external data files. These data files are written in JSON to allow for them to be altered or loaded by other tools, or if necessary, by hand. Provided is a basic command line interface to the library which can handle almost every step of the generation process. This is a simple example of a tool which can load and create the data files produced by the library, and utilise the library’s API. Providing the language-generation tools as a library allow for more complex tools to grow which implement and interact with the code.

The library format also allows for software which might have a need for ’conlangs’ to generate them directly and use them immediately, as opposed to generating one and then using the output files as input for the program. One such example could be in the case of a video game which requires a cohesive strategy for naming places, characters, flavour text etc. These entities could have their text generated on the fly, without any input from the game developer or writers. The *yod* library also allows the random number generator (which provides random values for all of the rest of the code) to be given a seed, meaning it will produce the same output every time. Thus, the objects in the aforementioned game could have different text every time the game was run, or the developer could choose a seed that gave output that suited them and keep that the same, without having to copy text from the output of the script into the game’s script and data files.

talk about structure

Chapter 2

Building Words

The 'naïve' method of building words is by taking an alphabet (for example, the Latin alphabet) and concatenating random characters until the sequence reaches a random length between a minimum and maximum. For example, given $min = 3$ and $max = 8$ we can generate the following example text:

Figure 2.1: Random letters from the Latin alphabet

```
tqk qzsyjla msmnix jvxx wug sysrh cuepg snyow ptjo bcek
arjdubw pfwpt nabgzk jmq taphh zewll dmpr uvpmx sfpfk uuo
bdm vnjbq hahuj wstq kohvma irn fott axdut rlgg tawz wsol
wigom psqwd tnv vlzgt lbcikk bof msmyg zkqgubb veht ukaznqn
ixp rppfj eqllnko uyyp aot uowtn icv fgypx cenawnk hypq
rruh eosgrf wmakeg hhweua gnbfh mkpzi ebtwbv cjwrxw ucky
kqezcm ucme wmrk khsya llzbeqw uxwivpp pbao gkzu pda txdp
iwl gkmfqn uxeupe atjxy vyul
```

Several problems can be immediately seen with this generated text. First, many of the words are difficult to pronounce and unrealistic with regards to their consonant clusters - for example, words like `jvxx` and `rppfj` are unlikely to exist in any natural languages. Generating words in this manner will also not produce very much variation in languages, as each letter has an equal probability to be picked.

We also quickly run into the problem of representing language in text. Written languages are based on spoken languages, so generating a written language first without basing it on a spoken one will lead to an unrealistic language. Furthermore, it is hard to say how our generated words are pronounced - one can apply English pronunciations to some of the words (for example `tawz` becomes /tɔːz/ and `axdut` becomes /ˈæks.dʌt/), but this results in a very Anglo-centric phonology, as we are biased to only use phonemes that exist in our own language

cite

while pronouncing unknown words.

Because of these problems, it is obvious that merely stringing together random characters with no thought towards pronunciation is insufficient when it comes to creating realistic and varied languages. Therefore it is necessary to first create a phonology on which to base all of our language's words.

2.1 Phonology

In its simplest form, a phonology is a list of every sound that is included in a language. English phonology, for example, contains around 24 consonants (with more or less depending on dialect) and anywhere between 7 and 14 vowels, again depending heavily on dialect. The phonology for the 'Received Pronunciation' dialect of English can be seen in 2.1 and 2.2.

cite (WALS)

Table 2.1: Consonant inventory in English phonology

		Labial	**Dental, Alveolar**	**Post-alveolar**	**Palatal**	**Velar**	**Glottal**
Nasal		m	n			ŋ	
Plosive, Affricate		p / b	t / d	t͡ʃ / d͡ʒ		k / g	
Fricative	**Sibilant**		s / z	ʃ / ʒ			
	Non-sibilant	f / v	θ / ð			x	h
Approximant			l	r	j	w	

Table 2.2: Vowel inventory in English phonology (Received Pronunciation)

	Front	Central	Back
Close	i / ɪ		u / ʊ
Mid	e	ɜ / ə	ɔ
Open	æ	ʌ	ɑ / ɒ

Other languages have different phonemic inventories, with varying numbers of consonants and vowels. Some, for example, have as few as 3 vowels, or as many as 84 or more consonants, depending on the method of counting[3]. A very large factor in the variety of languages generated is the phonology, as it restricts the type of sounds which often has a profound impact on how it is perceived. Therefore the first step towards a completed language should be a randomly generated, unique phonology.

The International Phonetic Alphabet (IPA) is an alphabet created by the International Phonetic Association for phonetic representation of speech and language[4]. It contains (or aims to contain) distinct symbols for each unique sound possible to create that is part of a language. As such, it provides a perfect way to convey the "end result" of generation process, since it can describe precisely how every word in the language is pronounced. The IPA also includes markers for syllable stress and other important factors which can be included in speech. In order to avoid the user from having to infer the pronunciation of a word from its orthography (how it is written), `yod` can produce an IPA transcription of its output, which shows exactly how to pronounce it without ambiguity.

However, the IPA can also be used as a basis for *creating* a phonology. It lists every sound possible in a language, so as a naïve approach would be to randomly choose sounds from this set to build words. This approach is similar to the approach we took before which involved choosing random letters from the Latin alphabet. However this time the words are being generated via sound directly rather than generating a written word, then trying to create a spoken representation of that.

The problems with taking the entire IPA as our set of possible sounds are twofold. First, it does not give us a unique phonology - instead the phonology is the same every single time, as its contents are not randomised. Second, if the phonology for the generated language contains *every* consonant and vowel from the IPA, then it will have many more phonemes than even the largest existing phonetic inventory. This is clearly unrealistic, and would be very difficult for anybody to learn. For artistic languages, this is a bit less of a concern but it is still preferable that the generated words be somewhat pronounceable and understandable, which gets less likely the more phonemes a speaker would have to discern from each other.

To create more realistic phonologies, the subset of sounds taken from the full phonetic alphabet can be restricted. If, for example, random phonemes from the set are formed into words of random length between 3 and 8, the following words could be produced:

Figure 2.2: Random subset of all phonemes

ɭθθɹ̝ çgd͡z ed͡zɻɭʋ ɭt̠̥͡ɹ̝gt̠̥͡ɹ̝χʊɻt̠̥͡ɹ̝ ɭʎ̝g̊ɹ̝ç d͡zħɖ͡ʐ ħħɻ̥ɹ̝eθ ed͡zçd͡z ʎ̥ʋd͡zeɭʋ ʊɭʊχħ

Although this method achieves variety of phonemes while keeping the total number of phonemes low - the phonology used to generate these words only contains 15 phonemes - a lot of the words are still unrealistic. Similar to figure 2.1, there are an abundance of large consonant clusters. Most words in the output lack a vowel and even those with vowels have extremely unrealistic consonant clusters.

Another problem with this generated output is the sequences /θθ/ and /ħħ/. The former contains adjacent voiceless dental fricatives, and the latter contains adjacent voiceless pharyngeal fricatives. Such sequences consisting of two of the same phonemes are unpronounceable and would most likely be realised as single instances of the phoneme (e.g. /θ/ and /ħ/). These problems show that this method of generation is still insufficiently realistic.

Is is important now to separate the phonology somewhat from the process

Figure 2.3: Shared properties of phonemes

- IPA symbol representation
- Sonority

of creating words. The problems just discussed are largely more to do with this word creation process, whereas there is still some work to do to improve the selection of individual phonemes for the phonology.

The first step is to create a distinction between the two different types of phonemes: consonants and vowels. Consonants and vowels have very different features, and play very different roles in the formation of words. Thus the phonology's set of consonants should be generated separately from its set of vowels.

Both consonants and vowels share the common features of phonemes shown in figure 2.3. Sonority is a numerical representation of how loud or sonorous the phoneme. Each phoneme also has a representative symbol in the International Phonetic Alphabet so they can be easily transcribed.

In addition to these shared properties, consonants also have the extra features shown in figure 2.4.

Figure 2.4: Unique properties of consonants

- Place of articulation
- Manner of articulation
- Phonation/Voicing
- Gemination

The place of articulation refers to the part of the vocal track that creates an obstruction of airflow. The manner of articulation describes *how* this obstruction affects the airflow - for example, ”stops” or ”plosives” completely block airflow, whereas ”sibilants” and ”fricatives” merely result in a turbulent airflow. The phonation of a consonant refers to whether, in producing the sound, the larynx affects the airflow through vibration of the vocal folds. Geminate consonants are consonants which are pronounced for a longer period of time, contrasting the usual shorter consonants. In 'stops' this is realised as a delayed release of the consonant, whereas other consonants with other types of articulation are merely prolonged.

Similarly to consonants, vowels can be distinguished by the features in figure 2.5.

Figure 2.5: Unique properties of vowels

1. Height
2. Backness
3. Roundedness
4. Length

A vowels characteristics are formed from the position of the tongue in the mouth and the rounding of the lips. The height of a vowel refers to the height of the tongue, and the backness is the forward/back position of the tongue. Roundedness is the state of the lips in pronouncing the vowel - either rounded or unrounded. This unrounded/rounded distinction can be seen in the difference between the two open mid-back vowels /ʌ/ and /ɔ/ (compare *shut* /ʃʌt/ and *short* /ʃɔːt/). The word *short* /ʃɔːt/ also shows a good example of a long vowel (although there is no long/short vowel distinction in most English dialects). cite?

In fact, both height and backness are defined by the formant frequency of the voice, which are produced by the tongue's position. A spoken vowel can be represented with two formants $F1 = height$ and $F2 = backness$. Roundness also has an effect on these formants, usually being seen as a decrease in $F2$ as well as a small decrease in $F1$.

It can be seen that both vowels and consonants have properties which relate to their length; these being consonants' gemination and vowels' length. Due to the similarities between these properties, they can be abstracted into the set of base phoneme properties as a 'Length' property. Thus we end up with our final set of properties being as follows:

Figure 2.6: Shared properties of phonemes, with length

- IPA symbol representation
- Sonority
- Length

This arrangement of properties lends itself extremely well to a simple data structure in `C#`. Consonants and vowels both translate trivially to classes, with their respective properties existing as enum members in those classes. For example, an enum `PlaceOfArticulation` can be created to represent the place of articulation of a consonants which would contain the set of values `Bilabial`, `Labiodental`, `Lingulabial`, `Dental` etc. Both the `Consonant` and `Vowel` classes are subclasses of a more general, abstract `Phoneme` class which contains only the properties that are shared between both of them. This allows the entire International Phonetic alphabet to be represented in code as a set of `Phonemes`.

It is more useful, however, to distinguish between the set of vowels and the set of consonants. Thus the two classes `ConsonantCollection` and `VowelCollection` can act as lists of their respective type of phoneme, with a `PhonemeCollection` being formed out of a union of the two sets of phonemes.

2.2 Syllables

2.3 Stress and Long/Geminate Phonemes

2.4 Words

Chapter 3

Orthography

Chapter 4

Grammar

4.1 Lexicon

4.2 Phrase Structure Grammar

Bibliography

[1] L. L. Zamenhof. Dr. Esperanto's International Language. 1887.

[2] Svend Nielsen. Per-country rates of Esperanto speakers. 2016. URL: `https://svendvnielsen.wordpress.com/2016/12/10/percountry-rates-of-esperanto-speakers/`.

[3] Georges Dumézil and Tevfik Esenç. Le verbe oubykh: études descriptives et comparatives. 1975.

[4] International Phonetic Association. Handbook of the International Phonetic Association. 1999.