The Haiku Generator

Saralin Zassman

Abstract

This report will discuss The Haiku Generator, a system that generates a seemingly infinite number of Haikus with just the press of a button.

As the world becomes more technologically advanced, it seems inevitable computers will begin to take over various forms of creativity. Many greeting card writers can easily be replaced by systems similar to the Joking Computer, or the Haiku Generator. The jokes or poems written for greeting cards are usually quite simple, following a set of rules depending on the occasion the card is being written for. This structure and simplicity can be replicated by a computer with little difficulty. I chose to build The Haiku Generator due to the versatility of haikus. The short poems generated by the system can be used in letters, text messages, greeting cards, and even as inspiration for longer poems or short stories.

The Haiku Generator is fed a series of pre-existing poems, and uses Markov chains to generate a large number of sentences from those poems. The text generated is then fed through a series of tests to determine which sequence of words are added to the final haiku. In this report, I will be going over the decisions I made to keep the haikus both interesting and as grammatically correct as possible. I will then discuss background material, various problems I encountered during the design process, the series of evaluation techniques I decided to use, and whether the system as a whole can be considered creative.

Introduction

A haiku is a Japanese form of poetry with a very distinct structure. The poem must consist of three phrases, the first being 5 syllables, the second being 7 and then the third being 5 again. Due to the well defined structure of a haiku, the end result always ends up having a characteristic sound or flow to it. For a system unable to detect intonation like the Haiku Generator, the final product can take on this characteristic sound as long as each line contains the correct number of syllables.

Haikus are often taught in elementary school because they are short, simple, and easy to write. Despite following a strict pattern, they can be quite versatile, and cover a large range of topics. For these reasons, I decided to create the Haiku Generator. The system creates a balance between the creativity of poetry, and the structure of a haiku. I plan to assess how meaningful or unique the final poems can be, given the structure restrictions put in by the system.

The report will begin with a summary of various related works, and how they compare to the Haiku Generator. I will then discuss some of the concepts I used as inspiration for my project, and how I went about using these concepts throughout my project. In the next section of the report, I will go over how I built the Haiku Generator, and various problems I dealt with during the implementation process. In the final section of the report, I will evaluate the project using the Turing Test and Colton's Creative Tripod, and then discuss some improvements I would make to the system as a whole.

Background

After reading the Poetic Machine, I discovered poets began using technology in the writing process in as early as the 1950s. I found it interesting how the first people to experiment with digital poetry were poets, and not programmers. Bailey and Funkhouser were solely interested in seeing how computers could assist with the poetry writing process, as opposed to having systems generate poems entirely on their own. Only in the early 2000s did the concept of computer generated poetry

start to become more substantial. The Poetic Machine discusses a program designed to create poems in Bengali. Unlike the Haiku Generator, the system creates a poem in collaboration with the user. Every time the user writes a line, the computer then responds with another in return. The report also describes how words are chosen to ensure that the final poem rhymes. Many forms of poetry such as Bengali are based on various rhyme schemes. However, because haikus do not usually rhyme, I decided not to implement any sort of rhyming scheme and further restrict what the system is able to produce.

From doing a quick search, I found a few haiku generation programs online. However, many of these programs follow a more fill-in-the-blank approach than a computationally creative one. In systems such as the Poem Generator[1], words are chosen by the user and then put together in the form of a haiku. The Interactive Haiku Generator[2] also follows a similar approach, where words fed into the system are randomly put into a series of pre-existing haiku templates. The user can then sort through and choose from the set of haikus produced. There is little creativity involved in software of this sort as the theme or message of the final poem is mainly determined by the user and the words they decide to input into the system.

Rather than having the Haiku Generator produce sentences from a set of templates, I felt it would be more interesting to use a variant of the Markov Chain algorithm discussed in the CO659 lecture titled Language, Stories and Conversation (Johnson 2020). In terms of text generation, a Markov Process determines how often certain words from the input text come after others, and then produces text based on those probabilities. Given the format restrictions of a haiku, I felt that using Markov Processes would give the software more flexibility, and a more diverse range of outputs as a result.

Design and Implementation

Initial plans

I decided to create the Haiku Generator with Java because it is the programming language I have the most experience with. Initially, I felt that creating a system to generate haikus would be quite straightforward given that many open-source libraries for counting syllables are available online.

Once I began the programming part of my project, I came across many obstacles that I did not anticipate for during the design process. Due to the simplicity of the program itself, I intended on building a more interactive software where the user is able to choose from a set of themes, and the system then outputs a haiku based on the theme selected. However, due to the many problems I faced when building what I thought was the foundation for the system, I decided to remove the interactive aspect of the project and focus on the quality of the haikus produced instead.

Open-source software

I decided to rely on existing software to count the number of syllables in each phrase, as well as for the text generation aspect of the system. This way, I would be able to focus more on improving the actual output of the software. Learning how to properly import an open-source library from GitHub was quite time consuming given that I had never done it before. However, the software I imported for counting syllables is definitely more sophisticated than any program I could create with the same function. After generating the first few haikus, I noticed that the library I imported would sometimes cause certain lines in the final haiku to contain the wrong number of syllables. At first, I tried to find which words were being labelled with the wrong syllable count and add them to the exceptions file.

---

[1] https://www.poem-generator.org.uk/haiku/

[2] http://www.languageisavirus.com/interactive-haiku-generator.php#.Xq5U2mhKg2w

However, this became quite tedious and time-consuming. Unlike stories which often follow a particular arc [2], many poems are more flexible in terms of structure and rhyme. In fact, a lot of famous haikus do not follow any specific syllable pattern. For these reasons, I am not too concerned with the occasional inconsistencies in syllable count.

Although we practiced creating a Markov chain algorithm in one of the CO659 practical classes, I felt using an open-source program would produce better results for the Haiku Generator. After much experimentation, I found that the final haikus sound the most poetic when the text generator is given only poems as input. When fed novels and short stories, the resulting haikus were quite awkward sounding, often containing more grammatical mistakes and less interesting phrases. The final input text file contains poems from a variety of different sources, including a collection of famous haikus, and a series of poems written by teens. I wanted the haikus generated to be lighthearted, and not too advanced in terms of vocabulary.

Generation of Text

The Markov Chain algorithm produces a large sequence of words from the text file provided. I found with 1200 words, the system is able to generate haikus at a reasonable rate without repeatedly running out of words to choose from. However, even if the program does run out of words to use, it is programmed to produce another set of phrases if need be. This way, no incomplete haikus are produced by the system.

In order to work properly, the text generator should only be given complete phrases as input. I noticed in the original text file, many of the poems have a capitalized title. Rather than removing all titles by hand, I made a method that creates a replica of the original text file but without any capitalized words or unnecessary punctuations. I then removed the rest of the titles, author names, certain numbers, and Roman numerals manually.

Building the Haiku

The text generated by the Markov Chain algorithm is then converted to a list data structure so that words and phrases can be removed easily. The system sorts through the elements of the list until the appropriate sequence of words is found for each line.

As mentioned earlier in this section, the system uses an external library to help with counting the number of syllables in each line. However, there is a lot more that goes into creating a haiku than just syllable count. The system follows a series of rules for each line in the haiku. Only sets of words beginning with a capital letter are allowed to make up the first line. This way, the line is more likely to sound like the beginning of a phrase. I found that without certain restrictions on the endings of each line, the haikus would sound quite awkward as the second or third line would often start without the phrase before it coming to an end. As a result, the system only selects phrases for the first two lines that either end with a comma or a semicolon, or do not end with a specific set of words. Some of these words include ‘the’, ‘and’, ‘if’ and ‘or’.

If the first or second line does not end with punctuation, the system will add a comma so that each line can sound more like its own complete phrase. However, the system does not do this to every single line as sometimes a phrase can stretch across more than one line. For example, in this untitled haiku by Masaoka Shiki, no single line makes up its own phrase.
‘After killing
a spider, how lonely I feel
in the cold of night!’

As a result, when the first or second word of a line is followed by a punctuation mark, the system does not add a comma to the end of the previous line. This reduces the occurrence of unnecessary commas as well as other

grammatical errors. For the third line, the system only picks phrases ending with a specific set of punctuation marks so that the last line resembles the end of a sentence.

Originally, each line in the haiku could be its own separate phrase from the text. However, the haikus produced would contain various grammatical errors due to the lack of cohesion between each line. As a result, I chose to add more restrictions to the program to try and reduce the number of awkward phrase transitions. I decided for the system to allow only a consecutive sequence of words to make up the last two lines. This guarantees a smooth transition between the second and the third line. When I tried to use this technique for the entire haiku, the results were less interesting and became quite repetitive. By enforcing too many restrictions, less phrases from the text were able to meet all of the requirements and be returned as output. A balance must be met for the system to produce versatile poems that are still properly structured and grammatical. After much experimentation, I realized that in order to keep the program interesting and unpredictable, I would have to allow for some degree of grammatical error to occur.

Results

In this section of the report, I will present a set of haikus produced by the Haiku Generator. After I briefly discuss these examples, I will then describe how the software works, and where I use open-source code within the program.

The following example shows a well written haiku. The poem itself is quite pleasant sounding, with each line transitioning smoothly into the next. Given that the original phrases are: 'She murmurs into his ear' and 'It might have graced a rosy bower', the poem is quite original. This is due to the Markov Chain text generator, as well as the selection process for each line.

'She murmurs into
force; He might have graced,
a rosy peace.'

The next example shows how the system is capable of producing haikus that rhyme. One of the original phrases used to create this haiku is: 'Of things discovered in the deep, Where only body's laid asleep.' Due to the input text file containing poems that rhyme, the Haiku Generator produces outputs that rhyme as well.

'Of this desolate,
cries in the deep, Where only,
body's laid asleep.'

Although the following haiku is quite poetic, it is identical to a part of the poem 'My House' by W.B. Yeats from the input file. Although the words are arranged in the form of a haiku, the phrases used are not original and can actually be considered a form of copyright. The system does not produce haikus like this very often, but I felt it was an important flaw to point out.

'An acre of stony
ground, Where the symbolic rose,
can break in flower.'

The next example demonstrates the creative potential of the system. Although the first line and the last two lines are from two completely different poems, the haiku carries the consistent theme of sadness or heartbreak throughout. Here, the word 'ships' suggests the idea of lost love, acting as more of a verb than a noun. Interestingly enough, the original sentence is about literal 'long grey ships'.

'Silence and love
ships. The sad, the lonely,
the insatiable.'

A lot of the time, poems do not have a clear or distinct interpretation. The message behind a poem is often dependent on how the reader interprets each phrase. In the following example, the first line is noticeably from a different sentence or poem. However, due to the restrictions I implemented on each line, the haiku still sounds somewhat grammatical. With poetry, there is more room for breaking certain grammatical conventions. This example shows

that as long as there exists some sort of structure to the poem itself, the reader can interpret the phrases however they see fit.
'What I can pretend,
not my yellow hair. I heard,
an old memory.'

In the following example, the haiku sounds quite pleasant, with each line effortlessly transitioning to the next. However, haikus do not usually contain specific character names or locations. Being only 3 lines, a haiku cannot describe certain people or places to the same extent that a longer poem can. As a result, some of the haikus generated end up sounding quite awkward or confusing due to the input file containing longer poems in addition to haikus. I felt that adding regular poems to the file would generate more interesting results than if the file were to only contain haikus. When I do come across poems like the following, I go into the text file and edit or delete the phrases that are responsible. This includes poems that contain dialogue, old-fashioned vocabulary, as well as the names of people and locations.
'Michael will unhook,
a bough overhead, And blow,
a little while.'

Overall, most of the haikus are original, interesting, and similar to regular poetry in terms of phrase structure and grammar. At the start of the implementation process, the system would output quite awkward sounding haikus due to each line being its own set of words from the text. However, by experimenting with different restrictions on the system and editing the input file itself, the final haikus rarely contain any obvious grammatical errors or confusing phrases.

Summary of Code

The 'Main' class generates a GUI to display the haikus produced by the system. The interface itself is quite simple and easy to use. Only one haiku is shown at a time, but the system generates another every time the button is pressed. An Action Listener detects when the button is pressed, and then calls on the 'HaikuAssembly' class to create another haiku.

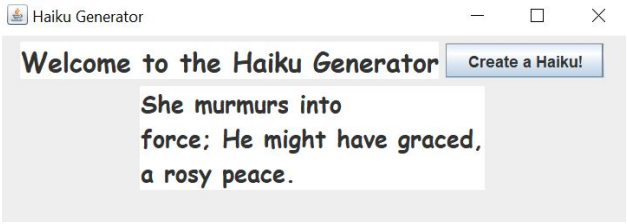

Figure 1 - The Haiku Generator GUI

The 'HaikuLine' class is responsible for generating each line of the haiku. A SyllableCounter object is created from the imported library, and used to count the number of syllables in each line. The criteria for each line is specified within this class.

The 'MarkovChain' class contains the open-source text generator. This class is called within the 'HaikuAssembly' class to output a large string of text based on the input text file. Then, the 'HaikuLine' class is called to return each line of the haiku. The 'HaikuAssembly' class puts the components of the haiku together, and returns the final product as a string.

Evaluation

In this section of the report, I will assess the Haiku Generator using a variant of the Turing Test, and SPECS Evaluation, one of the frameworks discussed in the CO659 lecture titled Evaluating Computational Creativity (Johnson 2020).

The Turing Test

I will be using an evaluation framework based on the Turing Test to determine the quality of the system and its artefacts. The Turing Test is used to determine whether a particular computer system can be considered 'intelligent'. A system passes the Turing Test if its artefacts are indistinguishable from their man-made counterparts. I will be using a variant of the Turing Test as the specifics of the original test are more suitable for a chatbot rather than a system like the Haiku Generator.

For the evaluation, each subject is shown five haiku pairings, and asked to pick which haiku is the computer generated one. The pairings are rearranged for each subject to prevent any particular pairing from affecting the data. I decided to use haikus by famous poets, as well as regular people and children to keep the test as fair as possible. However, I did decide to choose a set of more experimental haikus given that the system can be quite abstract. For example, the first poem in the following table would make for a better comparison than the second.

'Night; and once again, the while I wait for you, cold wind turns into rain.'
'After killing *a spider, how lonely I feel* *in the cold of night!'*

Figure 2 - Two Haikus by Masaoka Shiki

For the evaluation, I decided to choose a set of better quality haikus produced by the system. That way, the test acts as more of a reflection on what the Haiku Generator is capable of, rather than an analysis of the system as a whole.

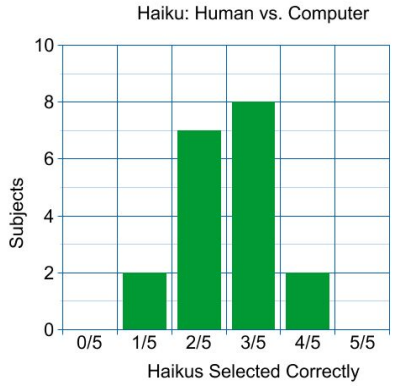

Figure 3 - Results of the Turing Test based Evaluation

The results indicate that the system is able to produce haikus indistinguishable from the work of a real poet. Around half of the computer generated haikus were mistaken for being human-written, with the majority of subjects scoring in the middle of the graph. This means that the haikus created by the system were not able to easily be picked out from the other haikus. In conclusion, the Haiku Generator is able to produce artefacts comparable in quality to poems written by a human.

SPECS Evaluation

Due to the previous test only assessing the system's best work, I wanted to use SPECS evaluation to look at the system as a whole. The survey asks participants to rate a series of haikus based on a set of questions provided. Each question tackles a particular characteristic of poetry, such as creativity or originality.

By using a range of different quality haikus, the evaluation is able to act as a representation of the system as a whole. I decided to include five consecutive haikus produced by the system to ensure the results are as fair as possible.

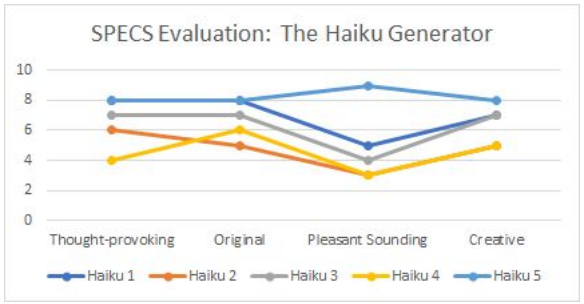

Figure 4 - Results of SPECS Evaluation

The data from the graph demonstrates the average rating from each participant, for the four characteristics in question. Haiku 5 scored well in every section of the survey. Being the only haiku rated as pleasant sounding, the other characteristics were likely influenced by this factor. In fact, the lower the score for the third column, the lower the ratings of the haiku in general. This makes sense as it is difficult to regard a poem as creative or meaningful if it is hard to understand, or grammatically incorrect. However, as I mentioned earlier, a poem is able to break various grammar rules and still be considered valuable or interesting. For example,

Haiku 2 was rated as thought-provoking, despite also being rated as awkward sounding or hard to read. This could be due to the phrases being less straightforward and more open for interpretation as a result.

Overall, the graph shows quite different results for each haiku. While some haikus were labelled as both original and creative, others were seen as uninteresting or hard to understand. This demonstrates how diverse the output of the system truly is. As we saw in the previous test, the Haiku Generator is capable of producing quite high quality artefacts. However, this evaluation shows how the majority of the haikus generated by the system cannot be put to the same standard as regular poetry.

In conclusion, the system as a whole cannot be considered creative. Although some of the haikus produced by the system are interesting and well-written, others can be quite ambiguous and difficult to understand. As I mentioned earlier, there is a degree of flexibility in poetry for breaking various grammar rules. However, if a haiku becomes confusing or hard to read as a result, the overall quality of the poem suffers. In order for the system to be considered creative, the majority of haikus produced need to be engaging and somewhat easy to follow.

Conclusion

The aim of the project was to make a computationally creative system that produces haikus. For the most part, I am quite pleased with the creative artefacts that the system is able to produce. Although the system does have some flaws in terms of consistency, I feel that the program itself is still quite impressive given that the haikus are produced by a computer system. Originally, many of the haikus were quite hard to read given that each line can come from a different part of the text file. After a great deal of experimentation, the haikus produced are much more coherent, with many even mistaken as being human-written.

However, due to not every haiku being up to this standard, the system as a whole cannot be considered creative.

As for practicality, as long as someone is able to sort through the system's output, the Haiku Generator can be used in many real world applications, such as greeting cards and letters. Even some of the poorly written haikus can be used as inspiration for poets and other writers. Although the system is only somewhat functional on its own, with user collaboration, the haikus produced can serve a multitude of different functions.

I plan to improve the overall quality of the haikus before adding any extensions to the system. It would be interesting to have the system evaluate its own results rather than rely on the reasoning of the user. However, this would require a greater understanding of machine learning and more programming experience in general. From a more realistic viewpoint, many of the poorly written haikus are caused by certain phrases in the input text file. By editing the file to strictly contain short phrases that make sense on their own, the haikus generated would be a lot more coherent. However, this would be quite time consuming and possibly reduce the versatility of the system. Once the haikus themselves are more consistent in quality, I plan to implement a more visually appealing GUI and have a relevant title be generated with each haiku. themes

Appendix 1 - Turing Test

Computer generated	Human-written
Among pale eyelids, heavy with tumult, their eyes, blue like the ice. Silence and love ships. The sad, the lonely, the insatiable. I should not squander, For fear of what we are, When gates clang shut. Of this desolate, cries in the deep, Where only, body's laid asleep. To be, with neither, purpose nor despair. Love was, not enough for us.	Night; and once again, the while I wait for you, cold wind turns into rain. - Masaoka Shiki The summer river: although there is a bridge, my horse goes through the water. - Yosa Buson Autumn moonlight- a worm digs silently into the chestnut. - Matsuo Basho Light of the moon Moves west, flowers' shadows Creep eastward. - Yosa Buson The crow has flown away: swaying in the evening sun, a leafless tree. - Natsume Soseki

Appendix 2 - SPEC Evaluation

1. Day after day, yet,
   never find a love Who loves
   another one.
2. Underneath each thought,
   images That have the new,
   still hides her horn.
3. Onward, homeward,
   yet so slowly we paced,
   When the night come.
4. His empty heart is
   sore. All's changed since I,
   hearing at twilight.
5. Sorrow dreaming, Went,
   walking with slow steps along,
   the unwieldy sun.

References

"Poem Generator." *Masterpiece Generator*, www.poem-generator.org.uk/haiku/. Accessed 3 May 2020.

"Interactive Haiku Generator." Language is a Virus, www.languageisavirus.com/interactive-haiku-generator.php#.Xq5ViWhKg2x. Accessed 3 May 2020.

Das, A. and Gambäck, B. Poetic Machine: Computational Creativity for Automatic Poetry Generation in Bengali. Available at: http://computationalcreativity.net/iccc2014/wp-content/uploads/2014/06/11.3_Das.pdf (Accessed: April 30, 2020).

m09/syllable-counter [Online]. Available at: https://github.com/m09/syllable-counter (Accessed April 20, 2020)

Jorente, P. (2020) Markov Chain Text Generator [Source code]. Available at: https://rosettacode.org/mw/index.php?title=Markov_chain_text_generator&printable=yes (Accessed April 20, 2020)

Johnson, C. (2020). Language, Stories and Conversations, lecture notes, Computational Creativity CO659 University of Kent, delivered January 29, 2020.

"Examples of Haiku Poems." Your Dictionary, examples.yourdictionary.com/examples-of-haiku-poems.html. Accessed 3 May 2020.

Jordanous, A. (2020). Evaluating Computational Creativity, lecture notes, Computational Creativity CO659 University of Kent, posted April 3, 2020.

"The Turing Test." Stanford Encyclopedia of Philosophy, plato.stanford.edu/entries/turing-test/. Accessed 3 May 2020.