Documentation of the computer generated novel Empathy

Alexe Luca Spataru

January 2020

Overall overview

The main idea behind the program is to read text, combine it then output a html with the newly generated novel. This is done with help of 5 python files and 2 html templates. The python code is purely functional. The templates are stored in the directory `/templates` and python files are in the main program's directory. In the `/lib` directory are all the books in the `.txt` format. In the `/output` are situated all generated text after program's call. The main part of the program is in those 5 python files, their purpose and content will be explained in the next sections.

read.py

The simple objective of this file is to read all texts from /lib directory and distribute them to other files. The texts are read in global variables which then are imported from other files. While reading the function slice() is used in order to cut the Project Gutenberg foreword and afterword so that no inappropriate text appears.

tools.py

This file contains a set of functions used in other files, like read.py the main purpose of this file is to be imported. Each function has its own task and overall they can be separated into text-fated and non-text-fated functions. Text-fated functions are numerous and require a separate section by themselves, while on the other hand, there are only 2 non-text-fated functions in `tools.py` : `ratio()` and `decide()`. `decide()` is just a simple 50/50 chance coin flip while `ratio()` outputs a number by multiplying the Feigenbaum constant $\delta = 4.6692016091$ with one of the five constants: $\pi = 3.1415926535, \varphi = 1.6180339887, e = 2.7182818284$, (Sierpinski constant) $\kappa = 2.5849881759$, (Second Feigenbaum constant) $\alpha = 2.5029078750$. This number is later used in order to decide the number of sentences to display in the output.

The text-fated functions can be classified in 4 categories, the ones that collect text, cut or modify text, combine text and the ones that identify text. The text collecting functions- `onlyWord(), onlyI(), onlyAlice(), collectHe(), collectShe(), collectI(), collectYou()`-always return a list of strings, where the strings are full sentences taken by a certain criteria. For example, the most important function `onlyWord(string, word)` takes a an argument a string, from where the data is collected, and a regular expression, in order to return a list of sentences that contain the word (hidden as a regular expression) specified in the argument. Other functions in this category are just instances or supplements to `onlyWord()`. `onlyI()` and `onlyAlice()` collect sentences that have the word 'I' and 'Alice', while `collectHe(), collectShe(), collectI(), collectYou()` collect sentences that contain 'He', 'She', 'I' and 'You', with a slight add-on. For example, the function `collectHe()` returns `onlyWord(string, 'He ') + [sheToHe(s) for s in onlyWord(string, 'She ')]`. That is, all sentences that contain a 'He ' and all sentences that contain a 'She ' converted into He. This is done in order to maximize the number of possibilities, thus to make the text more different after every generation.

The modifying text-fated functions constitute mainly of ones that replace a word with another, like `sheToHe()` used in the previous example. It should actually replace the whole subject in the sentence, but as that requires a more complex linguistic process it just replaces these words: She→He, she→he, herself→himself, her→his. In total there are 9 subject converting functions: `heToShe(),sheToHe(), ItoWe(),weToI(),heToI(),ItoHe(),ItoShe(),sheToI(),sheOrHeToYou()`. Other functions that modify text are those that cut it, it total there are 3. `clean()` just simply cleans newlinse and big spaces in order to not create any irregularities in the output. `strip()` removes first character if it is not a letter and changes it to uppercase if necessary, which is useful because in some instances text is collected with hyphens or lowercase. The `strip()` function is used only at the level of a sentence. `slice()` removes the Project Gutenberg foreword and afterword and is used at the level of full text, when the books are read in the file `read.py`.

The task of `mix()` and `conjunction()` is to combine text. The latter takes as input two lists of strings and a number n and outputs another list of strings of length n. The strings in the output are just jointed sentences of lists taken as arguments and combined with an " and ". These type of sentences are used in the second level of chapter Empathy. `mix()` takes as input a list of strings and a number n and outputs a set of strings of length n. Its main task is just to simply collect random n strings from the list takes as argument. The sentences are stored in a set in order to avoid repetitions.

In total there are 9 functions used to identify text. The most important is `search()` which returns true if there is an instance of the regular expression in the input string. Other functions are instances of this one, and their purpose can be detected by their name. They are: `hasName(),hasYou(),isChapter(), hasI(),hasShe(),hasHe(),hasWe()` and another function `matches()`, which is similar to search but it returns true if the given expression occurs at the beginning of the string.

colors.py

This file's main purpose is to create the last instance of the generated text. All functions in this file return a string, which are sentences joined together with html syntax for newline `<br>`. Strings are collected using functions from `tools.py` and stored into a list, then the number of sentences returned is decided by the ration between the length of the list divided by the number obtained from `ratio()`. The text which is build here include all chapters except Empathy, because this one requires a separate file which will be later discussed. For the first chapter-Solitude-the functions `invisible_alone()` and `I_am_Solitude()` are used. `invisible_alone` collects sentences that have the words, as the name suggests, 'alone' and 'invisible' from the book The Invisible Man. In total were collected 59 sentences. Because the main protagonist is a man, the sentences are converted from he to I in order to not assign any sex to the main protagonist -the 'I'- of the novel Empathy. `I_am_solitude()` returns a series of sentences that contain only 'I' and nothing else. There are quite a lot of them in The Invisible Man - 445 in total.

`interact()` and `unity()` are the basis of the second chapter Multitude. `interact()` collects sentences that contain the words speak, spoke and asked from the two books of Sherlock Holmes series. The number of sentences was narrowed so that they do not contain any name, to not include any other inappropriate character, and contain a You or He or She or We, to introduce the idea of multitude. `unity()` returns sentences that contain the words 'both' or 'together' to express the idea that a multitude can be one. The length of collected sentences in `interact()` and `unity()` is 106 and 100.

The last function `death()` used in the last chapter Death contains all sentences which contain the word 'death' from the books Frankenstein and Dracula. It receives an optional boolean argument which decides if the sentences returned have the subject an 'I' or a 'We'. The returned text is selected from a range of 107 sentences.

dice.py

Here, as in `colors.py`, the last instance of the text is formed. The text formed here is used in the chapter Empathy. In total there are 4 functions that do the work. Three of them are destined to generate the text for each level in the chapter, the other function `we()` is used if all the levels were passed. It returns the I sentences from chapter Solitude converted to a We. The other three function each return a tuple containing the text itself and a boolean which decides if the current level is passed. In each of the three, there is a similar process of building the return tuple. The text is built by collecting the sentences into two lists for each opponent and selecting randomly one sentences from the two lists. While the selection is done there are two variables that keep track of each 'opponent' occurrence. For example in the function `I_HeShe()` the counters are `countI` and `countHe` or `countI` and `countShe`. The counters will later be

used in order to decide the boolean's value by seeing if the difference between them is smaller than the gap. The gap is computed by the ratio of total 'battles', that is the number of times a sentence is selected from the two lists, and the number returned from the function `ratio()`. The number of battles is stored in the global variable `battles` and after each level it is incremented with the gap between the two counters. Initially `battles=(445/ratio())/2` because 445 is the number of I sentences in the chapter Solitude, and finally this is divided by two in order to minimize the novel's size.

write.py

If it was a C program the this file would be `main.c`. It does the actual writing and it is the file that should be called. It takes the text built in the previous files' functions and puts it into html using `jinja2`. It also, if required, converts the file into a pdf using `pdfkit`. It also, parses the command line arguments using `sys` and `getopt`. It contains 4 functions `empath(), render_empath(), write(), main()`. `empath()` does all the logic behind the levels in the chapter Empathy. It returns a 8-sized tuple, where the odd indexes are the text itself while even ones are the subchapter names. This function is used in `render_empath()` where the text is rendered using the html template `empath.html`. `write()` which takes as arguments an integer displaying number of copies to generate and a boolean displaying if the pdf convertion should be done takes the real latest version of the texts returned in the functions of `colors.py` and `render_empath()` and renders the final novel in the html format using the template `main.html`. The purpose of `main()` is just to simply parse the command line arguments and call `write` with the appropriate arguments. Besides that, this file has a global variable named `WE` which is modified in `empath()` if the We is reached. This global variable is also used in the call of the function `death()`.