2013

Visualizing the Novel

Clinton Mullins
Connecticut College, cmullins@conncoll.edu

ice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, and w
the use of a book,' thought Alice 'without pictures orconversation?' So she was considering in her own min
ell as she could, for the hot day made her feel very sleepy and stupid), whether the pleasure of making a dai
y-chain **The** girl would be worth the trouble of getting up andpicking the daisies, suddenly a White Rabbit wit
nk eyes ran close by her. There was nothing so VERY remarkable in that; nor did Alice think it so VERY muc
the way to hear the Rabbit say to itself, 'Oh dear! Oh dear! I shall be late!' (when she thought it over afterwa
occurred to her that she ought to have wondered at this, but at the time it all seemed quite natural); but whe
abbit actually, and looked at it, and then hurried on, Alice started to her feet, for it flashed across her mind th
he had never seen **Connecticut College Computer Science Dept** with either a waistbancoat-pocket, or not it a
atch to take out of it, and burning with curiosity, she ran across the field after it, and fortunately was just in t
see it pop down a large rabbit-hole under the hedge. In another moment down went Alice after it, never onc
onsidering how in the world she was to get out again. The rabbit-hole went straight on like a tunnel for some
d then dipped suddenly down, so suddenly that Alice had not a moment to think about stopping herself bef
he found herself falling down a very deep well. Either the well was very deep very shallow, or she fell very sl
r she had plenty of time as she went down to look about her **and** to wonder what was going to happen next.
rst, she tried to look down and make out what she was coming to, but it was too dark to see anything; then s
oked at the sides of the well, and noticed that they were filled with cupboards and book-shelves; here and th
he saw maps and pictures hung upon pegs. She took down a jar from one of the shelves as she passed; it w
elled 'ORANGE MARMALADE', but to her great disappointment it was empty: she did not like to drop the jar
ar of killing somebody, so managed to put it into one of the cupboards as she fell past it. 'Well!' thought Alic
erself, 'after such a fall as this, I shall think nothing of tumbling down stairs! How **Clint Mullins** all think me a
ome! Why, I wouldn't say anything about it, even if I fell off!' (Which was very likely true.) Down, down, down
ould the fall NEVER come to an end! 'I wonder how many miles I've fallen by this time?' she said aloud. 'I mu
e getting somewhere near the centre of the earth. Let me see: that would be four thousand miles down, I thin
or, you see, Alice had learnt several things of this sort in her lessons in theschoolroom, and though this was
VERY good opportunity for showing off her knowledge, as there was no one to listen to her, still it was good
ractice to say it over) '--yes, that's about the right distance--but then I wonder what Latitude or Longitude I've
?' (Alice had no idea what Latitude was, or Longitude either, but thought they were nice grand words to say.
resently she began again. 'I wonder if I shall fall right THROUGH the earth! How funny it'll seem to come out
mong the people that walk with their heads downward! The Antipathies, I think--' (she was rather glad there W
o one listening, this time, as it didn't sound at all the right word) '--but I shall have to ask them what the name
e country is, you know. Please, Ma'am, is this New Zealand or Australia?' (and she tried to curtsey as she
poke--fancy CURTSEYING as you're falling through the air! Do you think you could ofmanage it?) 'And what
norant little girl she'll think me for asking! No, it'll never do to ask: perhaps I shall see it written up somewhe
own, down, down. There was nothing else to do, so Alice soon began talking again. 'Dinah'll miss me very m
-night, I should think!' (Dinah was the cat.) 'I hope they'll remember her saucer of milk at tea-time. Dinah my
ear! I wish you were down here with me! There are no mice in the air, I'm afraid, but you might catch a bat, an
at's very like a mouse, you know. But do cats eat bats, I wonder?' And here Alice began to get rather sleepy,
ent on saying to **present**, in a dreamy sort of way, 'Do cats eat bats? Do cats eat bats?' and sometimes, 'Do b
t cats?' for, you see, as she couldn't answer either question, it didn't much matter which way she put it. She
at she was dozing off, and had just begun to dream that she was walking hand in hand with Dinah, and sayi
er very earnestly, 'Now, Dinah, tell me the truth: did you ever eat a bat?' when suddenly, thump! thump! dow
ame upon a heap of sticks and dry leaves, and the fall was over. Alice was not a bit hurt, and she jumped up
er feet in a moment: she looked up, but it was all dark overhead; before her was another long passage, and t
hite Rabbit was still in sight, hurrying down it. There was not a moment to be lost: away went Alice like the w
d was just in time to hear it say, as it turned a corner, 'Oh my ears and whiskers, how late it's getting!' She w
ose behind it when she turned the corner, but the Rabbit was no longer to be seen: she found herself in a lo
w hall, which was lit up by a row of lamps hanging from the roof.There were doors all round the hall, but the
ere all locked; and when Alice had been all the way down one side and up the other, trying every door, she
alked sadly down the middle, wondering how she was ever to get out again. Suddenly she came upon a little
ree-legged table, all made of solid glass; there was nothing on it except a tiny golden key, and Alice's first
ought was that it might belong to one of the doors of the hall; but, alas! either the locks were too large, or th
ey was too small, but at any rate it would not open any of them. However, on the second time round, she can
oon a low curtain she had not noticed before, and behind it was a little door about fifteen inches high: she tr
e little golden key in the lock, and to her great delight it fitted! Alice opened the door and found that it led int
all passage, not much larger than a rat-hole: she knelt down and looked along the passage kiss me please

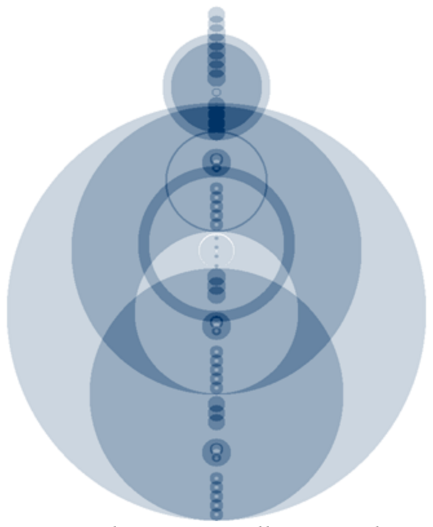
Visualizing the Novel

Thesis and code written by Clint Mullins
with Professor Bridget Baird as the project's faculty advisor.

1. Data Visualization

Humans are visual creatures, it is undeniable. Given the choice of viewing data in a spreadsheet or a graph most will choose to view the graph. While the spreadsheet might contain more specifics, the graph gives a quick and accurate synopsis of the contents, leaving the viewer with more time to think about the *implications* of the data as opposed to spending time trying to understand the data itself. This practice of visualizing data has been growing exponentially over the years, especially during the past decade or so. Information is being collected at such a high rate that visualization is becoming virtually necessary to keep data analysis up to speed with data collection. Some data are more easily visualized than others; for instance, if Boss Barbara wanted to answer the question, 'which employee had the most sales this quarter?' Barbara would tally up each employee's sales and represent them in a two dimensional (or three dimensional if she's a savvy Microsoft Excel user) bar graph. This is a simple and elegant solution to this problem. Now, instead of the Barbara having to sift through 230 employee accounts to find the largest number, she can instead look at a bar graph and instantly see that Johnson has clearly outsold his co-workers and deserves that raise and some praise. Good job, Johnson.

What if we wanted to visualize something more abstract? A man named Martin Wattenberg is fascinated with this problem: visualizing abstract data (1). In 2001 he set out to visualize music. He asked the question, "What does a song look like?" If one tried to visualize the song as whole, getting every detail, the visual would be too busy, and a viewer trying to pull information from it would quickly feel lost and confused. Much like Barbara's employee analysis, one needs to focus on a few details of the big picture. Mr. Wattenberg chose to focus on repetition in music. His program condensed songs down to a single note melody, looked for repetition in that melody (patterns of notes that appear more than once), and drew an arc from one instance of a pattern to the other instance. A skeptic could argue that this condensing of a song to single melodic line doesn't actually create a good representation, and that viewing this visual will be nothing like listening to the music. This is correct. However, the visuals help us better understand the music. An entire song cannot be encompassed within a static visual, but these elements of repetition are telling as


"As She Was" - Talking Heads

Repetition graphic by Mr. Wattenberg.

to the genre and style of a song before many other details enter the picture. It isn't a replacement for the song, but it is a visual aid, or a complement. Though we will discuss more examples of good data visualization in the related works section, it was when Martin Wattenberg spoke of his 'The Shape of Song' project (1) that this thesis was born as an idea to visualize the novel.

2. Visualizing the Novel

In Fall 2012 this broad idea of visualizing the novel had to become more than just an idea. What does it mean to actually visualize the novel? At first the greedy approach defined our goal; the visual should encompass the entirety of a novel and display everything about it. One should be able to look at this visual and say, "Ah, by your use of shape and color I see you've visualized the *updated* version of Huck Finn. Cheers." As usual, the greedy approach turned out to be a bad approach and the idea had to be refined. What does a story look like? Repetition in music was an interesting and informative concept to visualize because repetition is so important to music. Alone, it cannot tell you everything about a song, but it can deliver a good chunk of insight. What is important to a novel?

Characters. A story is composed of characters and their interactions. This may seem like a basic revelation, but it felt necessary to go back to basics in order to extract the essence of something. In order to visualize characters one must determine the essence of a character. Extracted attributes of a character could be anything, weight, race, gender, or height. Character height, while trivially interesting, wouldn't really tell us much about the story, which is what we are trying to get at. Through the scope of a character, how can we get to the story?

Emotions. If we could know how a character feels all throughout a novel we could see not just the events that occurred to that character or actions taken, but on a deeper level we could see really what is going on. By discarding the details of actual events and focusing on the emotions of characters, we could make a visual that doesn't replace the text, but complements it. A reader could see an overarching view of a character's condition throughout a text and try to pair that with their own understanding of the character. This visual could bring new ideas to the reader through the comparison of the reader's interpretation of the text versus how the data mining program interpreted character emotion. At this point one might be curious about the technical details of exactly how this could all happen. It is one thing to say 'extract character emotion' and another to actually do so in a meaningful way.

The method we use here is to create an emotional database of primary and secondary emotions.

The primary emotions we have chosen are [“Love”, “Joy”, “Anger”, “Sadness”, “Fear”] and each primary emotion has secondary emotions that help define the primary. We can then try to relate a character’s actions and descriptors to these categories. For instance, if ‘Megan smiles’, then we would try to relate 'smile' to all of the primary emotions. Love and Joy might relate highly to the word smile, whereas Anger, Sadness, and Fear might not. If we know all of Megan’s actions and descriptors throughout the text, and their relatedness to all our defined emotions, we could possibly map her emotional path throughout the novel. Now that we have started to represent emotion quantitatively (a process the humanities part of digital humanities might be skeptical about), we can begin to plan our visualization.

At each moment of a character being portrayed by new words, we have a relatedness of that word to each primary emotion. How can we visualize this data? There could be many ways, but we believe the most intuitive visual is the simplest: a two dimensional image with shapes flowing towards the right that grow and shrink. The x-axis represents the timeline of the novel. This means that the image, from left to right, is representing the first page to the last page. The y-axis is more complicated. In a general sense, each emotion is represented as a growing and shrinking shape. The shape spans the x-axis fully, but it grows and shrinks on the y-axis as the emotion becomes more or less relevant to the text. So using our example from above with Megan smiling, let's say in the next scene her pet Snookie dies. This makes Megan sad and so ‘Megan frowns.’ Megan smiling was the last moment our program saw. At this moment, the y-size of our love and joy shapes were large, while the rest were small. Then, when the program saw that Megan frowned, the love and joy categories shrunk greatly because the word frown is not closely associated with love and joy. In this same moment the category of sadness grows larger and fear/anger categories grow slightly larger as well with their relevance to the word frown being more prominent.

The program is written in Java and is fairly automatic. A user can put a text in the 'books' folder, change a string variable to that of their file name, and then run the program to receive a png image of their text. All code not mentioned in the related works section is new, written by Clint Mullins with the consultation of Professor Bridget Baird.

Using the techniques described above the program captures emotion and displays it in a readable manner. But this lofty goal of visualizing character emotion is not realized using all original code. To reach such heights one must stand on the shoulders of giants; luckily, those giants reside in the form of free and open source. In the next section we will explore some of these tools that allow the highly perched fruit of character emotion analyzing to be plucked. Afterwards, the methods section will explain exactly how this process works at each step; it will show how we use the established libraries and develop new methods to fit our goals. We will then show the visual model, how it can be generalized, and the current output with explanations of the good and bad results as well as what texts this program works best with. Finally we will conclude with our ideas of expansion, thanks, and works cited.

3. Related Works

This thesis can be broken into two distinct parts, data extraction and data visualization. Thus, we will speak of the related works in two parts as well. First we must extract meaning from text. This means taking any story, pulling the relevant data, and organizing that data into a format useful for visualization.

3.1 Semantic Meaning

There are many technologies to look at when it comes to extracting meaning from text. WordNet(2), for instance, is a heavy hitter in the world of extracting semantic meaning from words. What is WordNet? Back in 1985 a man named George Miller started a digital database of English words which were connected through certain relations (mostly synonyms). WordNet is still being updated today and continues to thrive in computational linguistics and natural language processing fields due to its open source nature; many programs such as JAWS(3) and WS4J(4) use the WordNet database. JAWS and WS4J are both libraries that use the WordNet database to draw connections through their own algorithms. All programs that try to extract semantic meaning must use a corpus of texts as a basis for their word relations. There are other technologies similar to those above that try to connect words via semantic relation. One of these other technologies is called DISCO and it is our semantic word program of choice. DISCO(5) can be used with many different types of databases similar to WordNet and it is very quick. The corpus (collection of written texts) we have chosen to work with is the British National Corpus (BNC). The BNC(6) was made in a different way than WordNet. The BNC site explains it best.

> "*The written part of the BNC (90%) includes, for example, extracts from regional and national newspapers, specialist periodicals and journals for all ages and interests, academic books and popular fiction, published and unpublished letters and*

memoranda, school and university essays, among many other kinds of text. The spoken part (10%) consists of orthographic transcriptions of unscripted informal conversations (recorded by volunteers selected from different age, region and social classes in a demographically balanced way) and spoken language collected in different contexts, ranging from formal business or government meetings to radio shows and phone-ins." - BNC (1)

DISCO was an ideal choice because it let us choose from many different corpuses. The BNC corpus appealed to us due to its collection from texts rather than from strictly handmade comparisons. Because we will be comparing words from stories it is good to know that the corpus was founded from stories, be they fictitious or not. While the idea behind using the BNC was a solid one, after testing the word relations for our purposes on many different programs and corpuses, it turns out they all perform somewhat similarly. In the end we stuck with our DISCO program running on the BNC and never looked back (after looking back several times, just to be sure).

3.2 Parsing Text

Relating words to one another is a big part of this project and is very important. But before we can relate important words, we need to know which words matter and which ones don't. For this task we needed a program that can tokenize words (a program that can read a sentence and use a best effort service to give each word a part of speech) and a program that would tell us which words in a sentence were related. This is not the same relation we are looking for from our DISCO program above, but instead we are looking for internal sentence 'dependencies'. If we looked at the sentence "George loves her." we can see that George loves someone, and that person is *her,* but how could a computer know that? This is where tokenizing and dependencies come into play. The Stanford Parser (a natural language parser) allows both of these operations and we have used those libraries in our program to do so. It is the slowest part of the process, leading to the program being not quite runnable in real time (takes a couple of hours to parse all the sentences of a novel), but it is the best we have currently. Let's look at our example sentence below as it is originally written, then tokenized, and then with dependencies all done through the Stanford Parser (7,8,9).

George loves her.

Nothing too special here. We have our normal sentence unfiltered and unchanged: au natural.

(ROOT (S (NP (NNP George)) (VP (VBZ loves) (NP (PRP her)))))

This tokenized sentence shows us that George is an NNP (proper noun), loves is a VP (verb), and 'her' is a noun. If we wanted to know all the characters in a sentence, looking at all nouns (proper or otherwise) is a good place to start. Now let's look at the dependencies:

[nsubj(loves-2, George-1), root(ROOT-0, loves-2), dobj(loves-2, her-3)]

Here we can see that there are three connections (or dependencies) in this sentence. The first, is 'George' and 'loves'. This is an important connection. We also see that 'loves' is deemed the root of the sentence in our second connection. Last but not least we can see that 'her' (or she) is love[d].

Using these two techniques we can extract specific data from text. I'm sure you can see how we could begin to form character profiles with the tools above, but we will go into that detail in the Methods section (stay tuned, it's coming up next!).

3.3 Topic Modeling

Topic Modeling is another avenue we explored to extract information from the text. Without getting too specific, topic modeling is the process of building abstract topics from a text by assuming words to belong to topics and trying to build up those topics with other words often used in conjunction with the abstract topic. Essentially, when running topic modeling software on a text, you get a list of topics with words that back up those topics. The implementation we were experimenting with was the UMass Mallet Topic Modeling software (10). Here is an example output for James Joyce's Ulysses.

t1 1 0.011 bread (34) tea (33) rev (25) butter (24) meat (23) eggs (20) milk (17) sugar (17) cream (16) kidney (16) waters (15) tribe (14) eating (14) burke (13) lump (13) hot (13) drank (12) ate (12) liver (12) librarian (11)

t1 2 0.873 project (80) work (40) works (39) human (32) terms (29) gutenberg (26) public (25) means (25) full (25) tap (24) male (22) form (22) part (21) host (21) person (21) free (20) copy (20) foundation (19) literary (19)

t1 3 0.065 man (348) bloom (346) time (293) good (257) stephen (205) day (195) night (189) thing (172) father (171) god (171) asked (162) life (157) make (156) long (156) poor (145) woman (140) don't (137) give (137) sir (132) young (130)

t1 4 0.035 eyes (226) hand (216) back (190) head (168) face (150) bloom (138) hat (119) round

(115) door (108) white (108) hair (98) open (90) left (86) stephen (86) hands (85) dark (85) air (83) miss (81) gold (74) light (74)

t1 5 0.016 street (232) john (118) mulligan (89) house (70) buck (70) dublin (68) mrs (64) irish (62) henry (55) lord (48) road (43) saint (38) bridge (34) north (32) hee (31) conmee (29) royal (29) city (29) father (29) company (28)

Though it looks a bit messy, what the Mallet software has given us is five topics (bolded) and the words that most back up these topics. The first topic seems to be based around all kinds of eating and drinking. The fourth topic looks to be about body images. As you can see topic modeling is a powerful tool, but we have decided not to incorporate it into this thesis. I discuss potential plans for topic modeling in the future works section, but as of now it remains a separate but powerful way to pull meaning from text. It is typically used to categorize a large base of papers into smaller sub-categories, and that did not prove useful for the particular problem at hand.

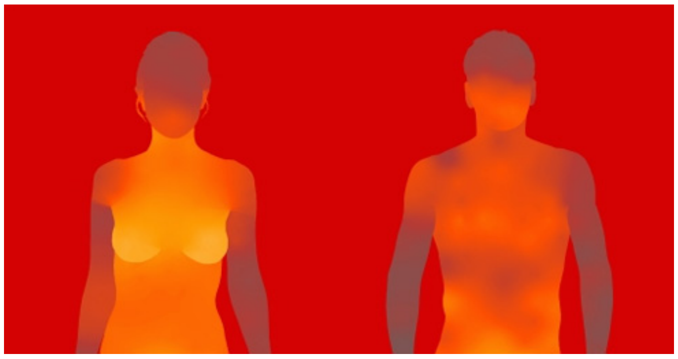
3.4 Other Visualizations

The second part of this thesis involves actually visualizing the story, which entails taking the organized data and presenting it graphically for a user to view and learn from. There are so many projects dealing with creative ways to visualize data concrete or otherwise that we could highlight here, but we will keep it brief and instead give links to sites devoted to this process such as visual.ly(11) and datavisualization.ch(12).

Flickrflow

The colors of the seasons.

As to some personal favorites of ours, we looked at Martin Wattenberg's work which was varied and fascinating. One of his most inspirational programs took daily photos of Massachusetts from Flickr for an entire year and then represented the 'colors of the seasons' in a circular graphic. The growing and shrinking of the seasons was a definite help in thinking about how to represent character emotion in this thesis. Another interesting visualization topic was where people liked to be touched by their lover(s). After asking many people in a survey how much they liked to be touched on each of their body parts, he created a kind of glowing graphic where the brightest glow


Fleshmap

Visit Mr. Wattenberg's website to see where humans like to be touched the most.

meant the highest response. The touch data was separated for men/women and the differences are sometimes obvious, but other times informative (though maybe just to us inexperienced youngsters). Beyond these two examples, there are many others. Another good resource to find great examples of visualizations is the webdesignerdepot(13) post.

As you can see there are many ways to visualize data. Some visualizations are very informative, others are less informative and more imaginative. It is hard to walk the line between getting the point of the data across and also making it appealing to view. This project aims to strike a balance in the middle.

4. Methods

We have touched upon what the program does and a little bit of how it does it, but here we will take an in depth look at exactly what is going on. Some of the information that the program extracts are not displayed in the visual, but I deem them important because future work on this project could definitely incorporate some, if not all, of the extracted information.

4.1 Gunning FOG Index

The first element we captured from the text was the Gunning FOG Index (14,15) rating of a novel. The Gunning FOG Index essentially gives a piece of text a reading level. That may seem like an unneeded step in this process, but in order to calculate this index one must find out many different variables from the text. The statistics found include total word count, average word length, total syllable count, average syllables/letters per word, number of complex words (defined as a word with more than 3 syllables), number of spoken words/sentences, etc. While these can hardly tell you the semantic content of what you are reading, we think they will eventually become important to the project in some way and, at the very least, would be interesting statistics to know. Certain authors are known for their run on sentences, while others are more terse. One could run this program on an author's oeuvre to see when certain patterns developed; they could try to see when their sentences became longer, if their vocabulary increased or simplified.

Before we further discuss all the data collected, we'd like to mention how our overall program sifts through the novel. Essentially, we have one for loop going through the novel character by character. Every time we hit a space, that's a word, every time we hit an ending punctuation, the sentence count goes up. There are flags that determine when words are a part of spoken dialog and other attributes. The novel is only read through once for all of the data collection. I did this in order to try and have a speedy program, but unfortunately the program as a whole is pretty slow mostly because parsers are understandably processing and referencing larges amounts of information.

4.2 Character Extraction

Character (protagonists, antagonists- not letters) extraction is done in a simple manner. While going through the novel in the process described above (character by character) we will often hit our sentence ending punctuation. At this point we do two things, though only the first is relevant now. This sentence first is sent to the Stanford Parser to be tokenized. This tokenizer is an impressive program that will tell us the parts of speech of each word. Not only does it tell us the parts of speech but it actually goes into much more detail by breaking the sentence into a hierarchy of sorts and gives more labels for words than I personally knew existed. For our purposes here (character extraction), we only care about one thing: characters. Take, for example, this sentence:

"Jerry went to the football game where he found his wife, Cheryl."

Here we have our example sentence. Clean and grammatically correct, our parser should have no trouble cutting through the mud to reach the golden character nuggets. Though there is some mystery as to whether Jerry first met his wife at this game or simply found the woman he had already married, we can let it slide. After all, this is just a sample sentence.

(ROOT (S (NP ***(NNP Jerry))*** *(VP (VBD went) (PP (TO to) (NP (DT the) (NN football) (NN game))) (SBAR (WHADVP (WRB where)) (S (NP (PRP he)) (VP (VBD found) (NP (NP (PRP$ his) (NN wife)) (, ,) (NP* ***(NNP Cheryl)****))))))) (. .)))*

Our new sentence is this tokenized version of our once nice and clean looking example from above. It now looks ugly and unreadable enough that, hey, maybe a computer will like it. As you can see, we have two bolded phrases above. The first bolded phrase is *(NNP Jerry)*, which means Jerry is a proper noun. The second bolded phrase is similar except that now we know Cheryl is a proper noun.

We can read this sentence and automatically assume these proper nouns are characters. How is that? We have rationalized this approach in three ways. The first reason we can do this is that important characters will be mentioned a lot. They should be the most prominent proper nouns in the book. At the end of our collection, we can sift through all of our ‘characters’ and throw out the ones with one or two mentions, only keeping the most frequently mentioned proper nouns.

The second rationalization is more philosophical. There could be a proper noun mentioned frequently that isn't a character, such as a location. Maybe a novel takes place in two separate locations with the protagonist flying back and forth, trying to choose between two lovers (email me for the rights to this story). In this case, New London, CT and London, England might be mentioned quite a bit. In fact, they might be mentioned so much and with such purpose that one could personify these locations. Maybe a militant humanist would scoff at this idea but we believe characters don't have to be living; as long as the author has put meaning and intent into an object, be it human, animal, or location, emotional content from this object could certainly be collected and used. If the data makes sense, use it, and if it doesn't, simply throw it out.

Third, if we want to map a character's emotions, this all works nice and well. However, there are certain words that will escape the direct character mapping of descriptors due to the use of multiple person pronouns such as we, they, etc. For instance, for the sentence, "Jerry, Cheryl, and Reese went to the movies and then *they* snuck into the theater." No specific character would have the word 'snuck' associated with them, even though they all snuck into the theater. With our system, 'they' becomes a character and all associations with 'they' are captured. This means 'snuck' will show up on our overall novel image, but not anyone's individual character image. While the issue of assigning 'snuck' to each of these characters is not yet solved in the current iteration of the program, if one does an emotional mapping of the entire novel, these large pronoun groups will be captured in the image.

When we detect a character we put it in the 'cast' hash table. A hash table stores keys with values and a program can very quickly (constant time) get to a value when given the key. Our cast hash table stores all the characters together and allows the ability for a quick look-up and change to each character object.

4.3 Character Shaping

Once we have detected a character, there are some useful things we'd like to know about it. As I mentioned above we have to keep track of every mention of the character. Every time we come across a mention of this character we add the index of that character mention (remember that we are going through the novel letter by letter) to the character's occurrences list. In this way we can keep track of specific mentions as well as the overall count of how many times they were mentioned. We then run into an annoying problem of pronouns. Take these sentences for example:

> "Mary ate about twenty grapes before putting them away. She knew that if she had eaten any more she would just get sick of that flavor. She is good at planning her life."

I'd just like to mention that I own the rights to all examples used in this thesis. We can look at this sentence and see, this is not just something about Mary, but this is all about Mary. However, our program, as far as we know, would only be looking for the name Mary and if we couldn't find that, we'd ignore the words in the sentence. This is where gender detection and pronoun mapping come into play.

4.4 Gender Detection

Gender detection is tough. Some novels will name their characters very androgynous names like 'Sam' leaving our programs scratching their virtual head pointers, not knowing what to do. To try and avoid this issue we have three different ways of detecting gender. Let's demonstrate them all in one sentence:

> "Mr. Hannifan was out with Jennifer when Sam showed up. She was putting mustard on her ice cream, daring to disturb the universe. Jennifer then...."

Our program would take Mr. Hannifan in as a character first. It would cut off the 'Mr.' but save it in the character class and set his gender to male because Mr. is a male honorific. If the program sees any of Miss, misses, Ms., Mr., Mister, etc it will assume the gender association with the honorific applies to the character. The second thing the program sees from this sentence is the name Jennifer. Jennifer has no honorific so the program then looks to a large list of male and female names. If one of the names matches, a gender is assigned. Then Sam comes along. Great. Well, in this case of androgynous names and no honorifics, our program simply counts the number of male/female pronouns stated directly after each mention of this name. Every time a character's name is mentioned they are put into a lastCharacterUni variable (if we have any idea of gender they are also put into either lastCharacterMale / lastCharacterFemale). We then count the number of male/female pronouns which in this case, would be +1 to female ("**Sam** showed up. **She** was..."). Without a definite gender definition these characters are defined based on the max value of male versus female pronoun counts. The system isn't perfect, but typically a name or honorific takes care of gender leaving less of this up to chance.

Side-note: While the real world is getting more progressive and gender is becoming less binary, I do feel a bit antique writing this program to shove characters into one definition of male or female. I did want to fully avoid gender as a metric but in the end, the need to know which pronouns refer to which character was too great to ignore. The world may not be binary, but the vast majority of pronouns surely are.

As I mentioned above, when characters have a gender definition they are put into lastCharacterMale/lastCharacterFemale variables so when any pronouns come along and we want to know who is being discussed, we can look back to the last mentioned characters of either gender and know with some certainty who 'he' or 'she' is. This system is not perfect, but making this perfect could be a thesis all in itself and one must draw the line somewhere. We think it works well enough for our purposes here.

4.5 Related Word Extraction

We mentioned earlier that once a complete sentence is read into the program we send it to the Stanford Parser for tokenizing. This tokenizing process is the first step in all the code explained above that revolves around character detection. Right after the tokenizing, we send the very same sentence into the Stanford Parser dependency detection algorithm. This algorithm takes the input of a sentence and gives an output of dependencies mapped to that sentence. For example, let's use our sentence from above with a little addition.

"Jerry went to the football game where he found his wife, Cheryl."

Once again, we have our simple sample sentence, nothing too flashy.

***[nsubj(went-2, Jerry-1)**, root(ROOT-0, went-2), det(game-6, the-4), nn(game-6, football-5), prep_to(went-2, game-6), advmod(found-9, where-7), **nsubj(found-9, he-8)**, advcl(went-2, found-9), poss(wife-11, his-10), dobj(found-9, wife-11), appos(wife-11, Cheryl-13)]*

And here is our sentence with dependencies mapped to each pair. Note that I have bolded two couplings. These are the pairs we look for: nsubj. There are many different pairs that this dependency detector can find. Here is an of example(4) of another dependency definition:

prep: prepositional modifier

A prepositional modifier of a verb, adjective, or noun is any prepositional phrase that serves to modify the meaning of the verb, adjective, noun, or even another preposition. In the collapsed representation, this is used only for prepositions with NP complements.

"I saw a cat in a hat" prep(cat, in)

"I saw a cat with a telescope" prep(saw, with)

Click the fourth link in the works cited and you can see an overview of all possible connections (there are many). Right now we only work with the nsubj which generally captures actions ("Bob played basketball.") and passive voice descriptions ("Carrie is sleeping."), perhaps in the future we will try to capture more elements such as adverbs but for now we have focused on these two methods.

Once we see a connection, such as nsubj, in our dependency sentence we look at the two words that are connected. Typically there is one pronoun or name and the other word is a descriptor of some sort. For instance: *nsubj(went-2, Jerry-1),* we can see that Jerry is our character and he 'went'. At this point the program searches for the name Jerry in our character hash table known as 'cast'. Because Jerry was definitely added in the tokenizing stage earlier, we will definitely find his name. Once we do find it, we add this word to his character's word hash table. Again, we use a hash table for quick look-up speeds. This word's hash table stores all of the words associated with a character. To get an idea of how this system works (one of the larger components of this project) I am going to show you the addWord() function for the Character class. Again, this is called when we find a word that is associated with a character.

```
public void addWord(String newWord, int wordInd, String pOS){
        if (!diction.containsKey(newWord)){
                diction.put(newWord, new Word(newWord, wordInd, pOS));
        }
        else{
                diction.get(newWord).addOccurence(wordInd);
        }
        Moment newMoment = new Moment(diction.get(newWord));
        emotionalRollercoaster.add(newMoment);
        //if (name.equals("clint") || name.equals("mullins)){
        Visualize.addToAllMoments(newMoment);
        //}
}
```

Our parameters include the string of the word itself (newWord), the index of that word in the text by character (wordInd) and the part of speech given to that word within the context of the sentence (pOS). If our word hash (called diction) contains the word we have, we simply add another occurrence to that word object, which is our character index parameter wordInd. However, if the word is not in our list we make a new word object.

Currently, for visualization, the character's word hash is not utilized. This may seem very counter intuitive because we have been explaining about adding to this character word hash and how to do so. This hash table stores a large amount of information about a character which could be used later on to expand upon this project, but as of now we use the Moment class to visualize. I do still believe the character hash will be important down the road because there are many different applications one could perform with that data (such as seeing the sum of a character rather than the per page approach we use), but because our image is linear, so is our data structure for building our image.

Notice that after the word is created there is also a new instance of the Moment class that receives the newWord string. This moment object is then added to the emotionalRollercoaster. The emotionalRollercoaster is a sequentially ordered linked list of all the moments in a character's life. But what is a Moment?

4.6 Moments / Emotion Spectrum / DISCO

Conceptually, a moment is a brief period of time in someone's life that is definable. If Samantha is entered into a triathlon and completes it over the course of seven hours, that is not a moment. But her conquest is composed of a string (not a CS string) of moments that started with her feeling determined and ended with her feeling accomplished. To us, and this program, a moment is simply a one word association. For example: "John kills a bug and then he washes his hands." There are two moments in here. First, John killed a bug. John killed. The second was John washed his hands. John washed. A word in a character hash is a culmination of all the times that a character was associated with that word, whereas a Moment is focused on one instance of association and, code-wise, it holds different information.

When a word is added to the character word hash, a separate moment is created and added to its series of moments. When a moment is created it tries to emotionally define itself. There are a few things one must know before emotionally defining a word makes any sense in a quantitative way.

Another class, called EmotionSpectrum, holds hardcoded and user defined emotions. These emotions are PrimaryEmotion objects. These objects are (for the sake of defining emotion) composed of words that build the concept of this emotion. Later we will look at the visual elements of PrimaryEmotions, but for now we will just describe the emotion words themselves. These words that build the PrimaryEmotions are called SecondaryEmotions. A PrimaryEmotion has one identifier such as "Joy". PrimaryEmotion "Joy" is supposed to capture all that joy is about. To do this we fill the PrimaryEmotion object "Joy" with similar emotions and other words that evoke that feeling. For instance, PrimaryEmotion Joy's secondary emotions are {"cheerful", "bright", "content", "beauty", "optimism","relief"}. EmotionSpectrum (the class mentioned at the beginning of this paragraph) holds all the primary emotions in a sort of virtual rainbow of emotion. The primary emotions we have chosen as of now are {"joy","anger","love","sadness","fear"}. If you can think of a time you felt an emotion that wasn't a combination of these, please let us know; or, as we will discuss later, download the code and change these for yourself. How? This list is completely editable and one could arrange any set of primary emotions and define define them with secondary emotions however they see fit.

Now that we have the knowledge of our EmotionSpectrum, we will discuss one more topic before moving back to our Moment class: this topic is semantic word relation. In the related works section we briefly touched upon DISCO, which is a semantic word relation program that uses very large networks of words to try and relate words to each other. This is the key ingredient of this program. Up until now, all we have is a character that is associated with many words and emotional categories which are composed of many words that build up primary emotions. Now we need a bridge to get from one side of the program to the other. DISCO allows users to get either first order similarity or second order similarity (or both). First order similarity measures the similarity between two words at face value. If one imagines a graph of all the words, connected via similarity, this value could be a distance between them.

(Joy <--> Frown) First order similarity (S1): **0.023083951**

As you can see there is a low correlation between these two words and rightfully so. First order similarity is a somewhat good metric but can be inaccurate, especially when dealing with words that aren't so obviously opposed. This is where second order similarity comes into play. DISCO can build a little collection of similar words to any given word. Second order similarity can test the similarity between any two of these collections.

Second order similarity tests on these two collections to get a value of

(Joy <--> Frown) Second order similarity (S1): **0.362491**.

This value is a lot higher, but all values of second order similarity are much higher than that of the first order similarity. Let's line those up with some other tests.

(Joy <--> Laugh)First order similarity (S1): **0.02839819**

(Joy <--> Laugh)Second order similarity (S2): **0.4619264**

As you can see there is a slightly better gain with the second order similarity, and although this is not always the case, it tends to be true. This is partly due to the fact the words that are related can be related by odd means. Let's take a look at the collections of similar words created by DISCO for one of our primary emotions ('joy') and a random word it could come across ('frown') to see where the issue could lie.

Joy's collection of similar words is:
happiness (0.1260), delight (0.1226), pleasure (0.1172), anger (0.0969), excitement (0.0913), pain (0.0842), satisfaction (0.0824), sadness (0.0816), sorrow (0.0742), enjoyment (0.0721), anguish (0.0711), fear (0.0703), pride (0.0699), tenderness (0.0696), tears (0.0693), gratitude (0.0674), longing (0.0674), warmth (0.0669), relief (0.0668),affection (0.0668)

Frown's collection of similar words is:
scowl (0.1052), smile (0.0887), flicker (0.0797), brow (0.0748), grin (0.0733), amusement (0.0731), brows (0.0706), glance (0.0637), sigh (0.0618), puzzlement (0.0610), nod (0.0606), shiver (0.0589), grimace (0.0582), gleam (0.0536), forehead (0.0527), silence (0.0523), expression (0.0519), shrug (0.0516), look (0.0512),gaze (0.0512)

Despite most of the connections making sense there are a few such as joy->sadness and frown->amusement that seem off. These kinds of connections are hard to get rid of because in some ways opposing emotions are related, especially in extremes. For example, love and hate are very related and if you don't believe that, watch any romantic movie or soap opera. We choose to use the second order similarity because we believe that overall it will work better, but a switch to first order similarity is a line of code away if need be. Some words will be much more related than others. For instance the word 'found' from our example above is rated below .01 on the second order similarity to all our emotions. Words that have no real relations to emotions or anything like 'was', 'is' etc will sometimes be picked up by the parser. In this case there is a bunch of zeros thrown into the average. At first we wanted to remove these as stop words because they would lower the amount of overall emotion there is in a section, however we realized that this type of inactive language, these boring words, were not a misrepresentation of emotion in the novel, but rather a representation of a lack of emotion. They lowered the average when they appeared because this section might be less important, and have less emotional impact by design.

Now that we have established our library of emotions in EmotionSpectrum, our character's word relation techniques, and the DISCO word semantic similarity program, we can finish explaining the Moment class. When moments receive a word they measure semantic similarity between the word and all of the different emotions, using an algorithm that does the following:

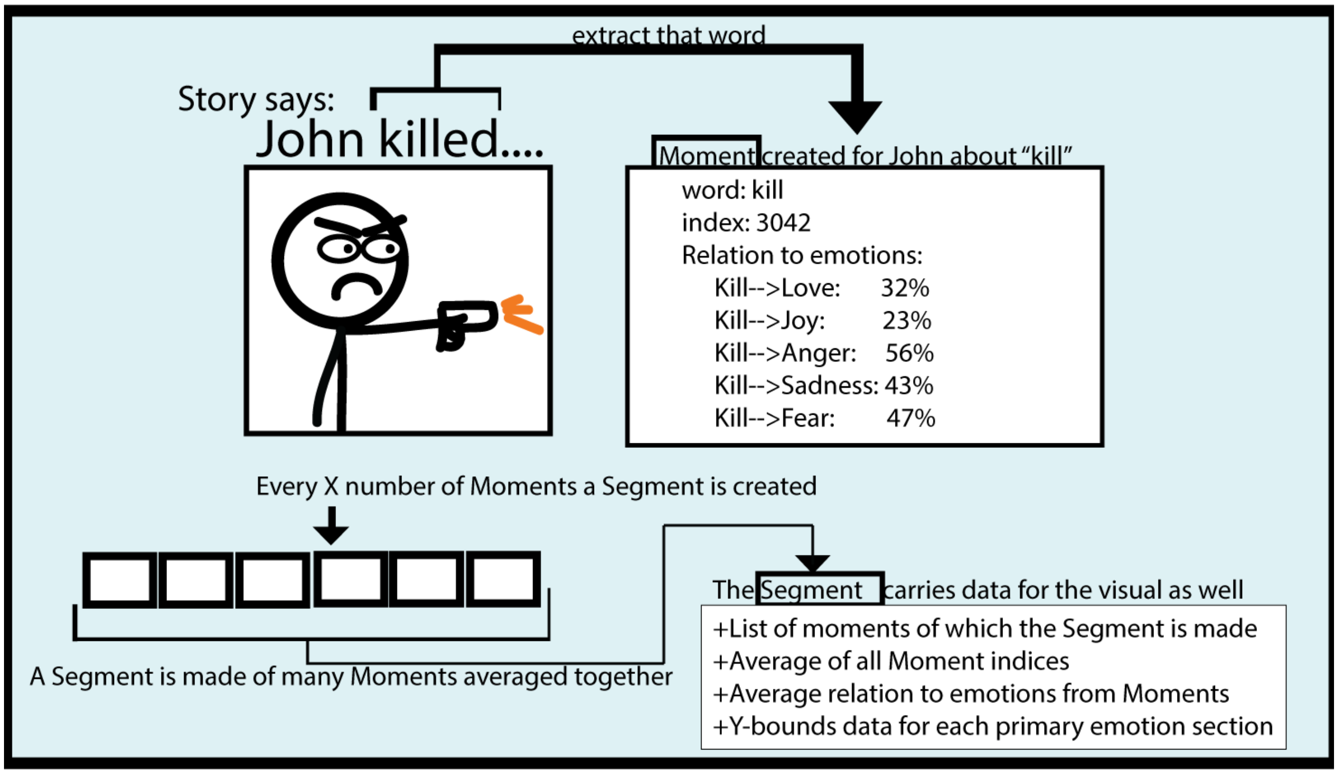
```
for each PrimaryEmotion
        relatedness[primary] = secondOrderSimilarity(word, primary)
                for each SecondaryEmotion
                relatedness+= secondOrderSimilarity(word, secondary)/# of secondaries
                relatedness[primary]/2
return allRelations
```

The allRelations variable here represents an instance of a class called MomentEmotions, which essentially holds the relatedness of all the emotions to that particular word. Each moment object holds an instance of the MomentEmotion class and it is how a moment is defined emotionally using MomentEmotion and MomentEmoteSec which represent a string(primary/secondary emotion such as joy) and value (relatedness to the Moment's word) for primary emotions and secondary emotions, respectively.

I'll explain again from the bottom up to hopefully make things more clear. There is one MomentEmotion for each primary emotion the user has hard coded. For every hard coded and defined secondary emotion within that primary emotion there is a MomentEmoteSec object in the primary emotion representative MomentEmotion. These objects lie in the Moment object, and their only purpose outside of visualization is to hold the relatedness value that DISCO assigned them in reference to a word. This word, along with the MomentEmotion objects, are stored in a Moment. After a moment object receives these MomentEmotion objects (full of MomentEmoteSec objects) it puts them in a sorted priority queue , with the most related at the head. There is an example below.

When we began running tests with Moments and displaying them we found that Moments were a bit unwieldy. Especially if there is an error in identifying the correct implied emotion from the author there could be a passage that seems calm and then has a spike in the amount of fear detected due to one odd word. This problem is due to the fact that we are taking words at face value. To fix this spiking issue we created a class called Segment. A Segment class gathers up a fixed number (let's say 6) of sequential Moment objects and essentially averages them with each other. This smooths out what would have been outliers into a smaller and gradual change. This variable is adjustable though, if it is made too large the image flattens. If it is one, the image has many sporadic jumps, but can look more interesting.

Below is a flowchart that tries to quickly and simply explain this process. If any confusion remains please take a look at the graphic. I'd recommend taking a look regardless because it's pretty nice-looking.


extract that word
Story says:
John killed....
Moment created for John about "kill"
word: kill
index: 3042
Relation to emotions:
Kill-->Love: 32%
Kill-->Joy: 23%
Kill-->Anger: 56%
Kill-->Sadness: 43%
Kill-->Fear: 47%
Every X number of Moments a Segment is created
A Segment is made of many Moments averaged together
The Segment carries data for the visual as well
+List of moments of which the Segment is made
+Average of all Moment indices
+Average relation to emotions from Moments
+Y-bounds data for each primary emotion section

Here we illustrate the process in a wonderful infographic

5. The Visual

Up to this point we have discussed how the data is extracted and we've made mention of the visualization but with no real specifics. This section looks to change that; here we will define our visual model and show many examples to explain how it works.

5.1 Concept Process

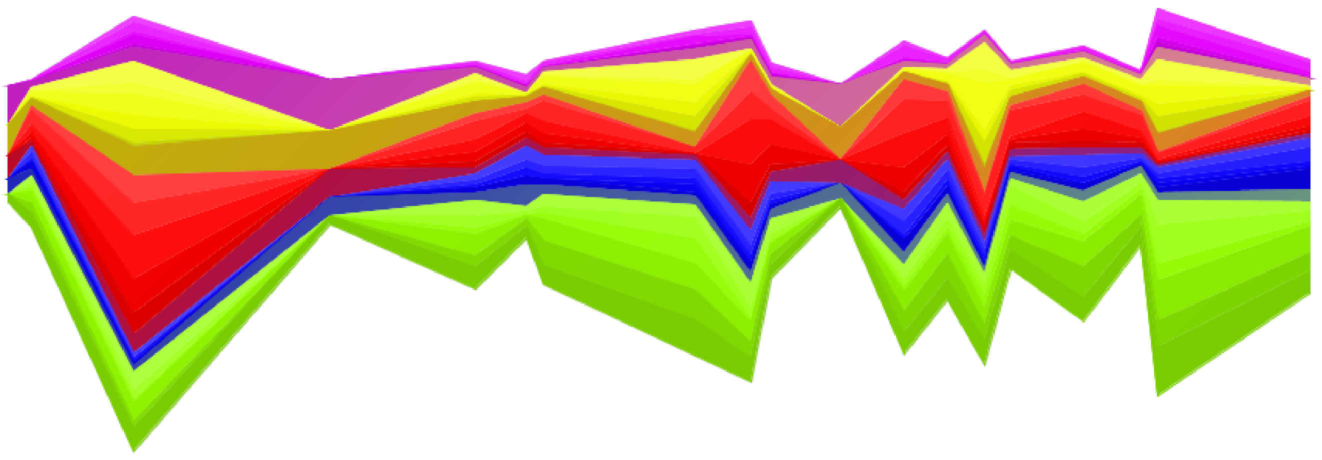
As discussed in the introduction, the visual was originally going to represent the entire novel, trying to bring in as many aspects as possible to most accurately show what is happening in some sort of 'spiritual' sense, trying to show the soul of the novel. After doing research on typical visualizing projects there seemed to be two extremes. On one end we had visual info-graphics that very explicitly showed facts with accompanying visuals to make the data more accessible and on the other end we had extremely abstract visuals (like the old Windows Media Player music visualizers that resembled fireworks) that looked nice but showed no real data correlation unless you were very well versed in the inner workings of the program. In the beginning we wanted to create a visual that could somehow encompass the strengths from both ends of this spectrum; highly detailed yet imaginative and beautiful. When trying to straddle two extremes one tends to end up in the middle, and we believe that is where we reside.

Looking past the grand notion of visualizing every detail of the novel we started to focus on certain elements, the most important ones being novel mood and character mood because they seemed, as far as abstract data to extract from a novel goes, reachable. With this in mind we started working with character extraction, gender detection, relevant character diction extraction, and relating these characters at every moment in the novel to some degree of different emotions to try and find the dynamic mood. Once this was nearing completion the question kept coming up, 'how to visualize this in an appealing way, but also an informative one?' As far as informative we knew there were a few details that must be shown. The first is the character/novel mood; this is what we have been trying to get at. The second is how that mood changes overtime to shape the novel/character. After reading a novel or short story, the viewer should be able to look at the related image and understand most of it ("Ah, this is where Harry's wife died, and I can see his Joy decreased, his Love remained constant, and his Sadness, Fear, and Anger rose."). That is the ideal scenario. Even more idealistically the viewer might see a character's emotion chart and disagree with the evaluation, then they could try to figure out why the program detected so much fear from a seemingly confident character. They could then look back and see that the character had more bravado than bravery. Of course, those would be the ideal results from this image, so how could we display that?

Because we have two items that we want to display, we keep the image two dimensional, there is no gain in adding needless complexity to an image. The two items are character/novel emotion and time (index in novel). As westerners typically read left to right, let's have time flow from left to right on the x-axis. This leaves us the y axis to represent emotion at each point in time. We could horizontally group a series of bar graphs at each moment with relatedness factors on the y-axis; it would be quite ugly and boring to look at, though informative. What if we took those bar graphs and instead of lining them up horizontally, we stacked them at each moment? This is slightly more intriguing but still a bit clunky and somehow not as informative as we would like.

At this point we connected the bar graphs to each other. Now instead of one set of emotions after another, there was a fluid transition from one to the next, leaving a sort of trail that represented a climb or decline in emotional relatedness in each emotion from one moment to the next. The image was getting simultaneously more informative and more interesting to look at. We decided then to weight each emotion to make the 'winners' stand out more. When taking each emotion at face value there was too little difference between emotional relatedness (this is something we get into in deficiencies/future work section). To counteract this we decided on a weight factor to beef up our winners and shrink our losers. This is where we have landed today, and though we have more we'd like to do with this visual described in the future works section, the current iteration looks nice. It is explained in our next section.

5.2 Current Visual Model



Top to Bottom: Joy, Love, Anger, Sadness, Fear

Here is an example of an outputted image. As you can see it is as we've described in the previous section. Colors flow from left to right allowing the reader to see where there are spikes of certain emotions along the text. This text is a short horror story called Laurel: The Girl I Loved(16) categories, secondary emotions are also represented as shades of the primary color. These secondary emotions, all together, hold the same weight as the primary emotion in terms of relevance to the word at hand.

To create this image we gather all the sequential segments and use the data within them to plot boundaries for each separate shape (a shape is one emotion). Each of the MomentEmotion class objects (remember, these objects hold one emotion's relatedness to a word- this word is from the Moment class, the moment class has one MomentEmotion for each primary emotion) are put into a priority queue based on relatedness to their word. So if I were to have a Moment of the word 'yelled' the priority queue of MomentEmotions could be:

Anger - 0.5342123
Love - 0.3432458
Fear - 0.3242433
Joy - 0.2323444
Sadness - 0.1442526

You can see that the most related emotion, anger, is at the top. This ordering is important for what we do next. As we've said earlier, in each Moment object, each emotion is given y-bounds to show how relevant that emotion is to the character/novel at that moment. We tackle this task with a class called MomentEmoteSpace. This class holds two important values; the first is the final y-bounds for an emotion, and the number of points it has. The number of points an emotion has greatly influences how large this emotion's y-space will be. We assign these points in a simple manner, and this is where our sorted priority queue comes into play. I should note that this all happens exclusively in the segment class. Remember that the segment class is essentially many Moments averaged together. The number of Moments in a segment is completely under user control and can be as low as one (effectively saying that a Moment *is* a segment).

We iterate through our sorted priority queue of MomentEmotions with a set number of points to hand out. Let's say for this example that the number of start points is equal to the number of primary emotions (five). We give our most related MomentEmotion five points, then subtract one from the start points to make it four, give that many points to the next most related, subtract one, etc. so in our example above it would look like this:

Anger - 0.5342123	P-5
Love - 0.3432458	P-4
Fear - 0.3242433	P-3
Joy - 0.2323444	P-2

Sadness - 0.1442526	P-1

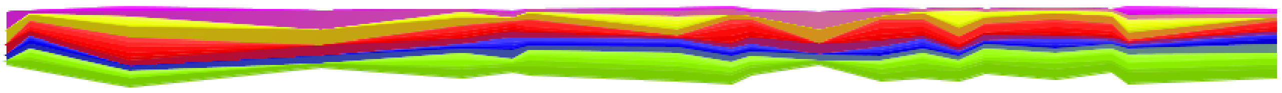
Now that we know the total number of points, we divide our total y-height by this number to get the value of one point in terms of y-space. In our example this total point value is 15, and if we had a total y-space of 1500, each point would be worth 100 y-units. If this were the end of the algorithm each of these emotions would receive the following y-space:

Anger - 0.5342123	P-5	y-space: 500
Love - 0.3432458	P-4	y-space: 400
Fear - 0.3242433	P-3	y-space: 300
Joy - 0.2323444	P-2	y-space: 200
Sadness - 0.1442526	P-1	y-space:100

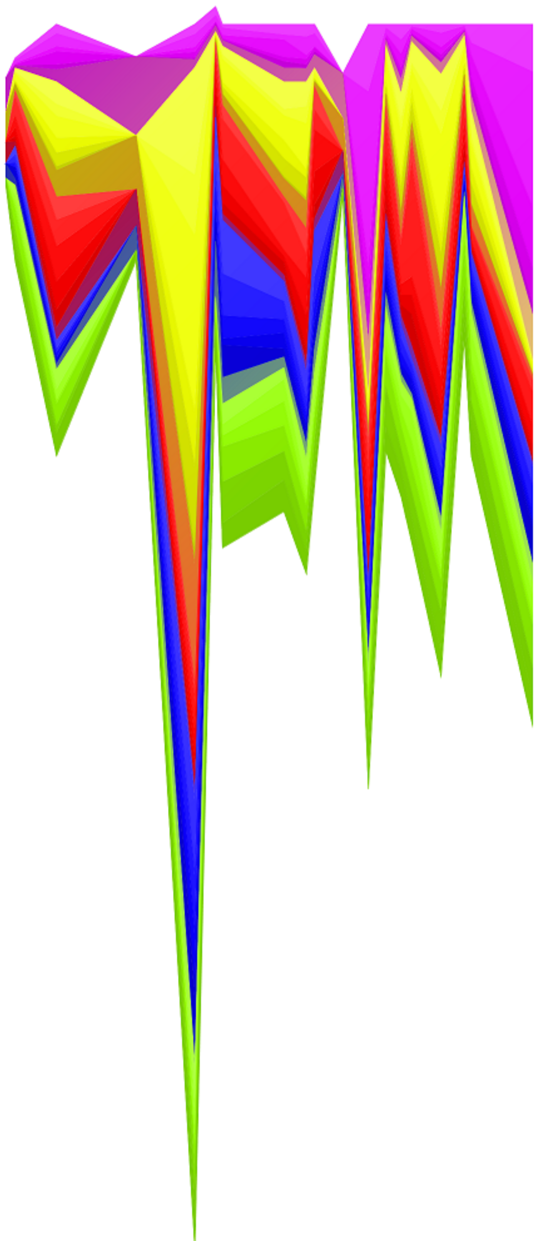
It is a bit like a race where times don't matter, and only place order does. This is when we made the relatedness value come into play. These points are then changed on a per emotion basis based on their relatedness value.

```
        int threshold = 10
        double pointHeightChange = getMomEmoteVal(newBounderies[i].getEmotion())/.05;
        if (pointHeightChange<1){ //if it is less than one, it is equal to one
pointHeightChange=1;
        }
        else if (pointHeightChange>threshold){
                pointHeightChange=threshold-.01;
        }
        curBottomNewYTop = (int)((newBounderies[i].getTop()+(((onePointHeight/(threshold-
pointHeightChange))* tempPoints))));
```

Here is the code for evaluating the changing of point height values. First we get our emotion's point value and divide it by .05. This typically nets us a value between 1 and a threshold value. If our value is not between 1 and threshold, we force it to be, for the sake of the image bounds. Then to calculate the distance (I've bolded this code) we first subtract pointHeightChange from threshold. The pointHeightChange value has a direct relation with the actual relatedness value. We introduce a threshold variable and use it to invert this relation, making the larger relatedness values produce smaller pointHeightChange values. We divide onePointHeight (which represents the size of one y-space chunk, 100 from the example above) by this inverted pointHeightChange and multiply it by the number of points this emotion received. The threshold value is somewhat important. If it is too low, the image stretches way beyond the y-limits of our frame. But if the threshold is too large, the image flattens out to a boring and uninformative image. Based on the relatedness we usually receive (between 0.0 and 0.5), 10 seemed to be a good medium, but this value can change on a per story basis.

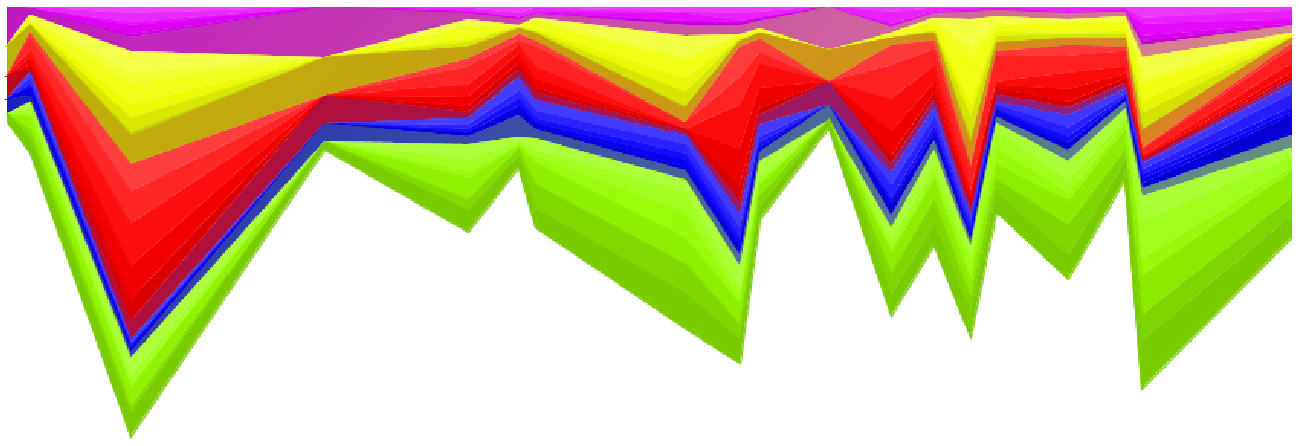


Above is an example of when the threshold is too large. The values become too small to matter and shrink away. On the other hand if the threshold is too small, the values become enormous and fly off the graphic as seen on this page. Note this image is shrunken but not distorted, its length is equal to that of the image above.



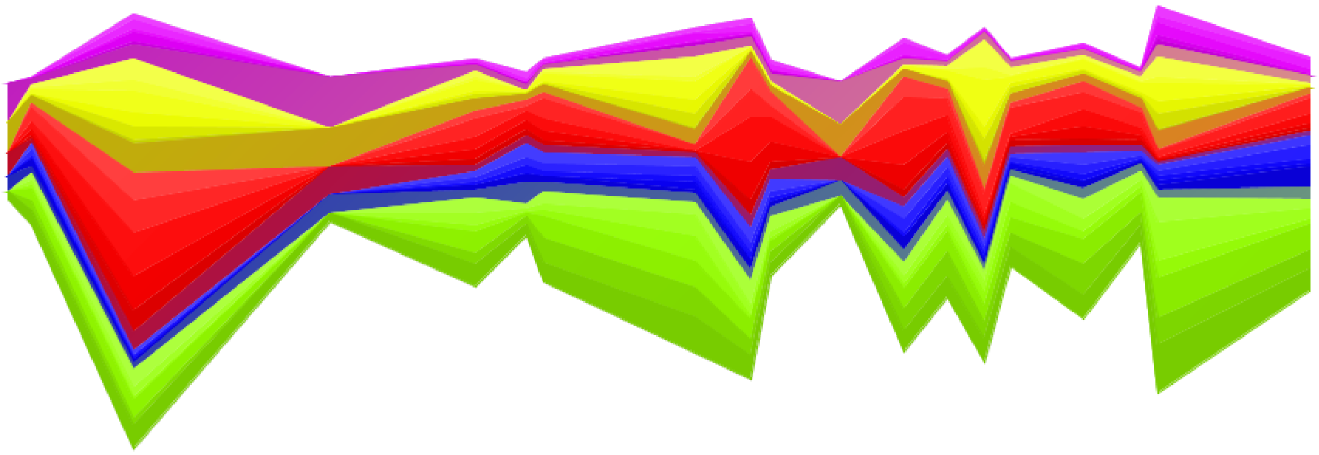
To recap this is a more simple manner what is happening exactly, for each segment, each emotion is given a number of points based of how relatively related it is to a word compared to the other emotions. These points directly map to a chunk of y-space. Then we change the size of these chunks for each emotion individually based on their absolute correlation regardless of the other emotions and give them X number of chunks based on their points. In the code above the MomentEmoteSpace (held in newBounderies array) is not only receiving a size of y-space, but an actual y-coordinate. The first of the emotions has its y-top set at 0, then gets its y-space from the algorithm we just described and through that it gets its y-bottom. The next emotion's y-top is set as the previous emotion's y-bottom, it then gets its y-space, adds it to its y-top to get its y-bottom. This process continues for all of the emotions in a segment until they all have their y-bounds. After they are all assigned their y-bounds we do two more things.

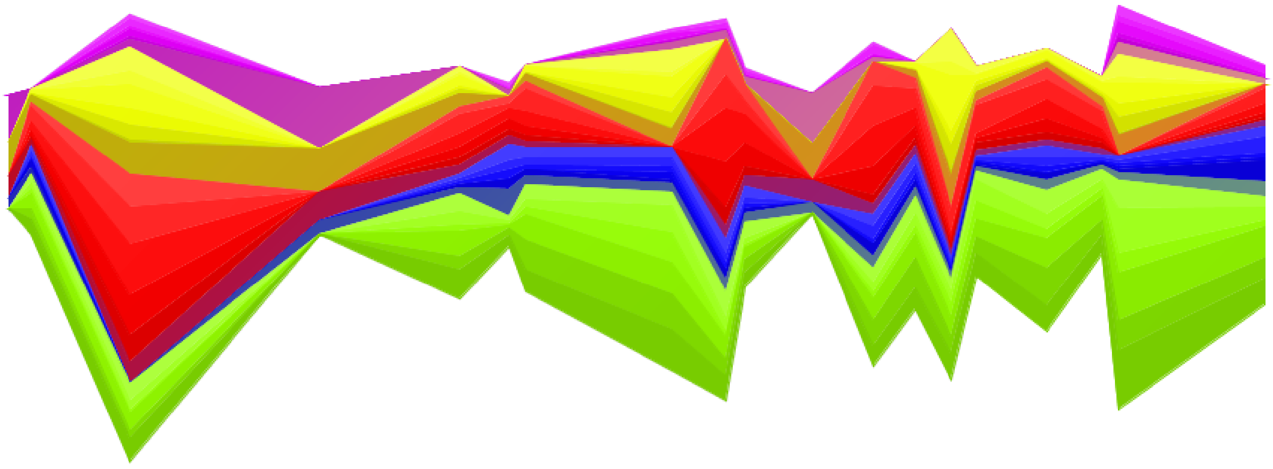
If we left this alone there would be a flat surface on one side where everything grows from. Instead, we want a more flowing image. So when we check for each emotion's relatedness we also find the most related and set it to be the 'anchor'. The anchor influences the entire image towards its original y-bounds. The algorithm accomplishes this by finding the difference between where the current y-bounds of the anchor are and where the default position of the anchor is supposed to be and using this we get a travellAll value. We then go through each of the y-bounds and move them the travelAll distance. This keeps the image from having any flat sides. Below is an image that shows the sample from above except the image has not undergone this shifting process.

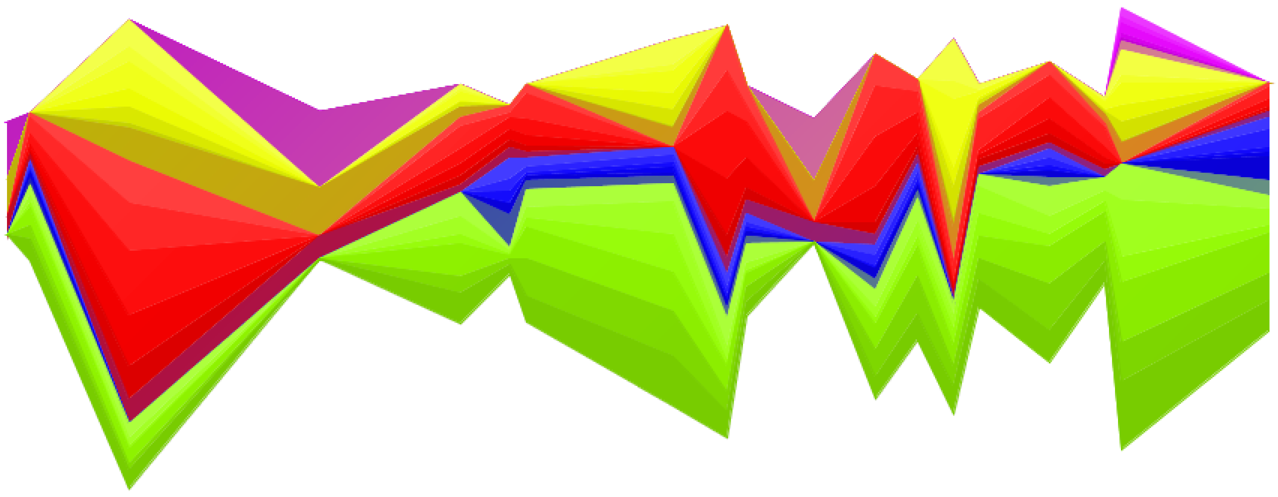


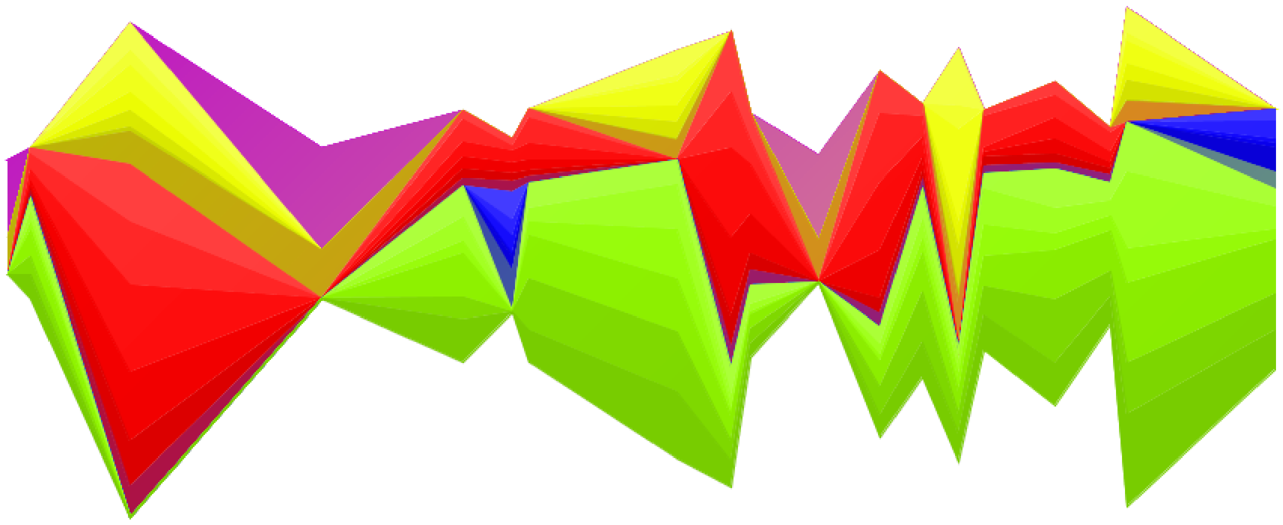
After giving each primary emotion its permanent space we can now assign the secondary emotions their y-bounds within the primary emotion. To do this we simply add up all of their relevance values to get a total value. Then we get each secondary emotion's percentage of that total, and give each secondary that percentage of the primary emotion's current space. It is a simple, yet effective way to portray the secondary involvement.

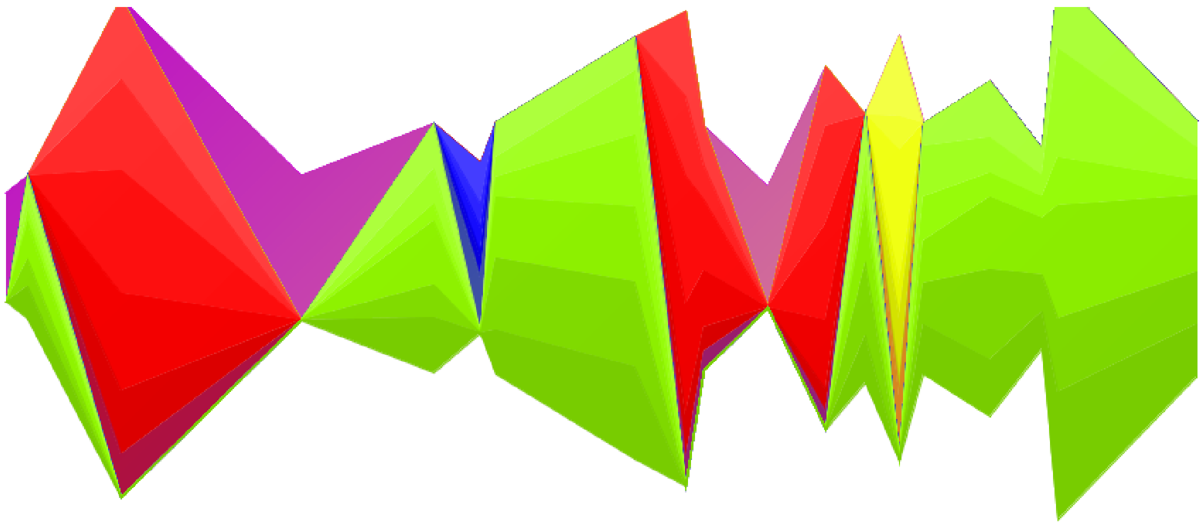
Something we spoke about earlier: the number of start points for the MomentEmotions, is a variable that can have an interesting effect on how the program works. Typically we give a number of points that equals the number of emotions. So, for instance, if we had five emotions, we'd have five start points. This means the most relevant emotion gets five points, then next gets four, the next gets three, and this continues until the last emotion gets one point. If we give more start points than the number of the emotions we have, nothing really changes. There are slight variations but the image looks mostly the same. However, if we reduce the start points to below the number of emotions we get the cutting off of the least relevant emotions at each step. I will now show five images, all with the same data but each time there is a difference in the number of start points for the algorithm. Watch for the smallest colors to fall out first, and then the larger ones until, in the final image, only the highest relevant color is left.









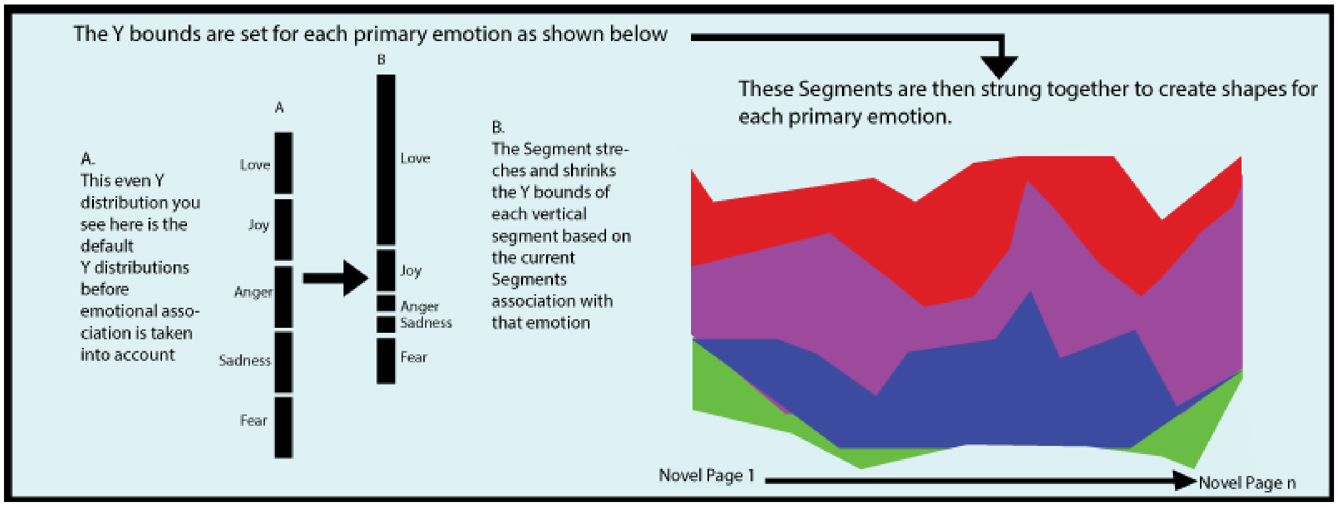


At every segment, there is only a certain number of primary emotions visible. This kind of process can do some neat things. For instance if you look at our last image here, only the most prominent emotion from each segment is shown. Also note that the image grows vertically a bit because only the most relevant primary is taking up any y-space, and it has the biggest chunks delivered to it from the algorithm. This image shows that fear (green) is the prominent emotion in this story, followed by anger (red).

5.3 Java2D

The way we chose to display this data was through the java2D libraries. First, we create an array of shapes the size of the number of different emotions. We went through each segment and built each shape. So for each segment we went through the y-top values and added them to the shape, when we reached the end we went through each segment again. This time we went through them backwards and capture the y-bottom values because we were continuing to build each shape from where we left off with the y-top values, which is all the way at the right.

Once the shape was completed we colored it in with the appropriate values. As for the colors in the image, these are user chosen. The user chooses a hue from 1-360 and the image will use each hue to paint the appropriate color for the shapes. If we have the primaryOnly boolean set to true, each shape has one color and no secondary information is shown (though secondary relevance values still influence each primary emotion's space). If we set this boolean to false, the primary emotion is made up of however many secondary emotions and the colors of these secondary emotions are automatically assigned based on the primary hue given. Saturation and brightness values are dragged from one side to another to give an even looking gradient across the primary color.

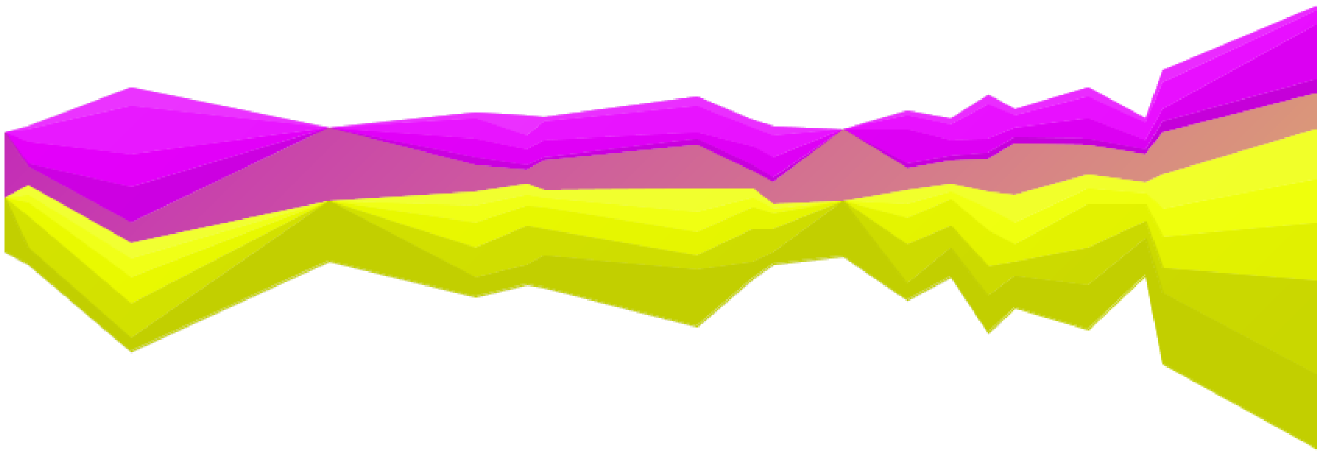

The Y bounds are set for each primary emotion as shown below
A
A.
This even Y distribution you see here is the default Y distributions before emotional association is taken into account
Love
Joy
Anger
Sadness
Fear
B
Love
Joy
Anger
Sadness
Fear
B.
The Segment stretches and shrinks the Y bounds of each vertical segment based on the current Segments association with that emotion
These Segments are then strung together to create shapes for each primary emotion.
Novel Page 1
Novel Page n

Flowchart that explains the Y distribution.

5.4 Generalizing the Visual Model

We certainly have spoken a lot about characters, moods, emotions, etc. But this thesis contains a model that can be generalized. What does this mean? The visual model we have created uses primary and secondary emotions to check the relatedness of emotions to words we have related to characters. These primary/secondary emotions, as we've stated before, are coded into the program and they can be easily changed by anyone with a text editor. We wanted to try and find the mood of characters in the text. However, if one wanted to try and test other relatedness categories it's possible to do so.

For instance if we could replace of our primary and secondary emotions with two primary *categories:* male and female. Then we could build up these categories with secondary categories that helped define what exactly male and female mean. If we wanted to view this through a traditional lens of gender we could file [“strong”, “tough”, “determined”, “hard”, “cold”] under the male primary and [“compassion”, “beauty”, “love”, “giving”, “soft”] under the female primary. We could then run this same program on a novel or short story to try and answer the question of “Does this text/character inherently relate to the traditional male or female stereotype?”. Here is a quick example of running the program on the horror story from above. Here the female section is pink, while the male section is yellow.



As you can clearly see, the program sees more of a male influence.

Even further in this direction, all the visualization needs in order to function is a set of moments (whatever a user wants to define as a moment in their text) with each Moment containing primary/secondary categories that have values from 0-1. This means that with more tweaking of the current program or the inclusion of another program to gather this data, the visual could still be a useful aid in understand or displaying results. Using the DISCO word semantic relation one could pull the relation data to whatever category the user chose and display the data using this visualization code. For example, it could give a student who is about to read a paper for a class a better idea of what that paper is about dynamically so they will know what to pay extra attention to and what could possible be skimmed.

This is all a bit hypothetical, but in the academic world a hypothesis is half the battle. Even if this code in itself is not used, the graphic has an informative look that doesn't just give an overall approximation such as 'This article is about [30% autocracies, 40% oligarchies, 30% dictatorships]' but it could give a step by step read out of the content of an article. Sometimes a straight summation isn't very useful: this project could be individualized to deal with other issues similar to this example.

6. Results

Here we will show what we've done, analyze how correct our results are, and try to determine if there is merit to this project. We thought the best way to test this is to give the program several different short texts that we could analyze ourselves and contrast the computer's results with our own. For these texts we will try to choose stories that tend to lean in an emotional direction or have obvious drastic changes. We will visualize a character from each and also the text as a whole to see how they differ and if they are accurate.

6.1 Text One - Eternally

While scouring the internet for a good text to represent love I came across a Christian short story website. Religious stories generally have some unfiltered love baked into them. I found a short story called “Eternally”(8) that is about a young couple about to be married. The woman, Sarah, accidentally loses the engagement ring a few days before the wedding and stays home from work to find it. When the husband, Jack, shows up at work to surprise Sarah and take her out to work, she is missing. He goes to their house to confront her and finds her crying. He is reminded of how his mother never told his father that she was sick until it was almost too late; he begins to get worried. She tells him she lost the ring and has been looking for it; he comforts her and she finds the ring. She asks him to put it on her finger, he does, they are in love, the end. For this text we will look at images from the story as a whole, and then individually at Jack and Sarah. The first image shows only the primary emotions while the second shows the secondaries as well. This is how we will presenting images in this section. We spoke earlier about avoiding unnecessary complication in images and while we do like the look of the secondary emotions being represented, one could argue that it is unnecessary and distracting. Here we give the reader a choice of both. Remember that the colors still represent [“joy”, “love”, “anger”, “sadness”, “fear”] and they have that same order in the image from top to bottom.

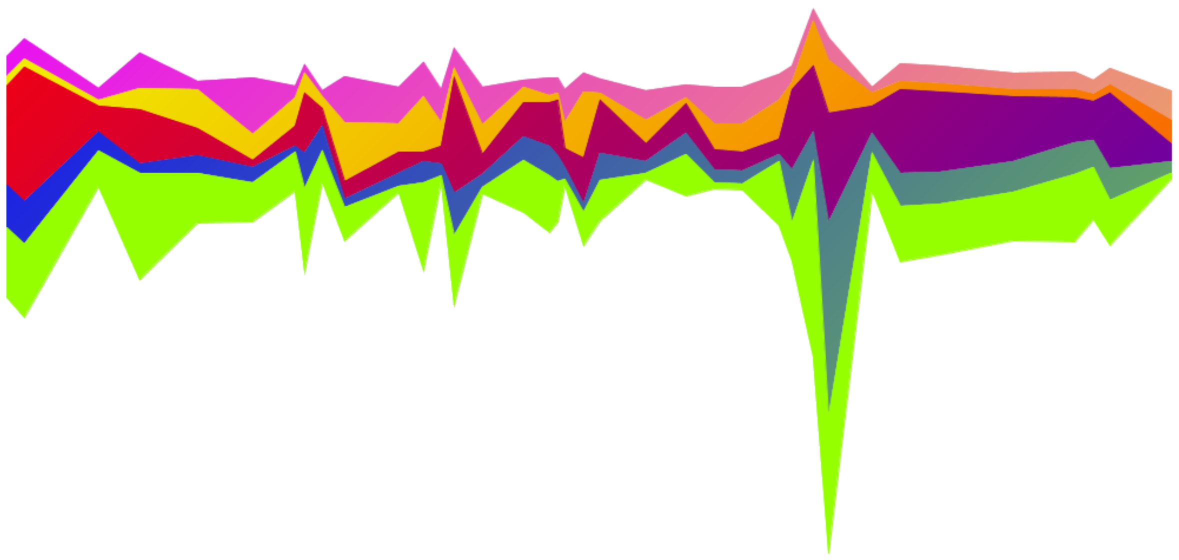
Joy

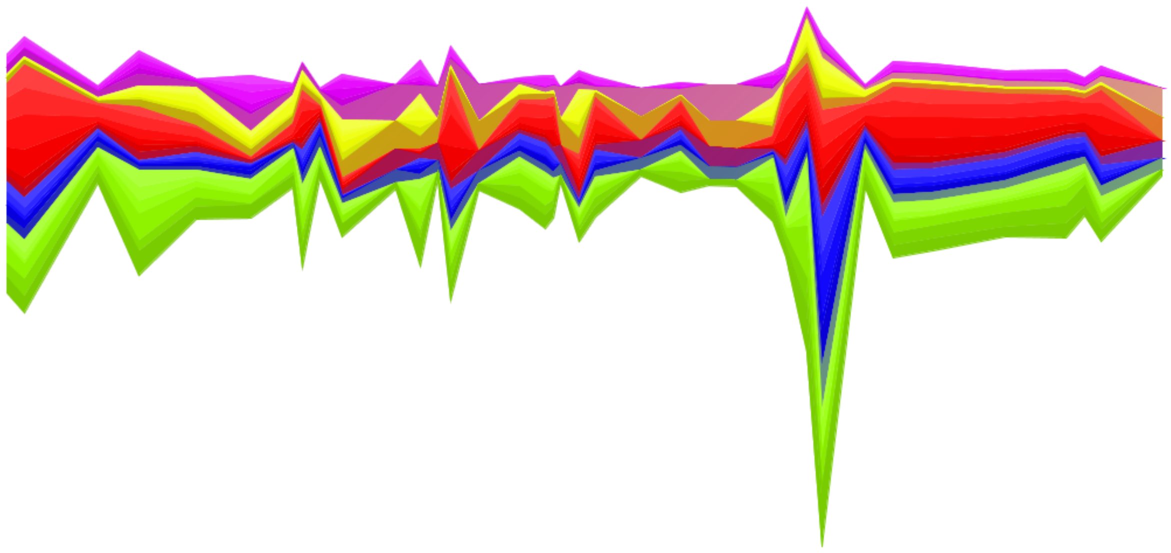
Love

Anger

Sadness

Fear

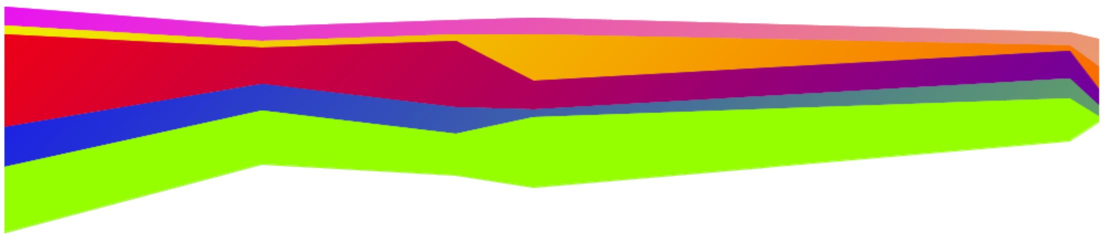




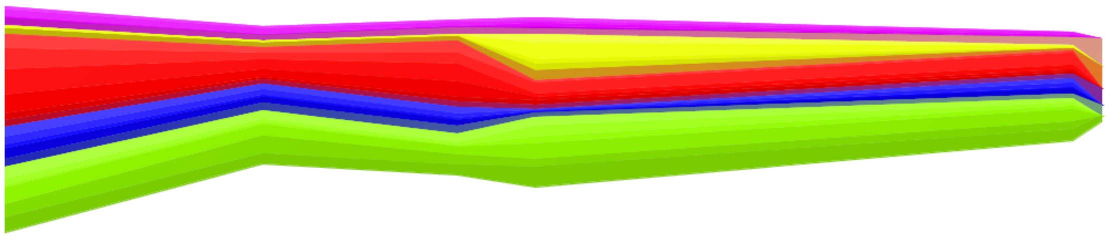
.

The beginning starts off with an abundance of anger, fear, then a bit of sadness. This sounds about right as Sarah was very mad at herself for losing the ring, but fearful of Jack's reception of the news. As the story progress the anger calms down save for a few spikes and most emotions seem to be somewhat equal; this might be due to her sister calming her down and speaking of multiple subjects at once, including that jack loves her and he won't be mad. At one point, about three quarters in, there is a huge spike in sadness, anger, and fear. This could be the moment when Jack walks into the house and sees Sarah crying and silent. He starts to speak of his mother who kept secrets and how it pained his father. This conversation lasts almost until the end of the story which could explain the large section of anger, fear, and sadness after the spike. These feelings all subside at the end: this is where Jack and Sarah find the ring and they are both happy and embrace, exchanging the words “in sickness and in health.”

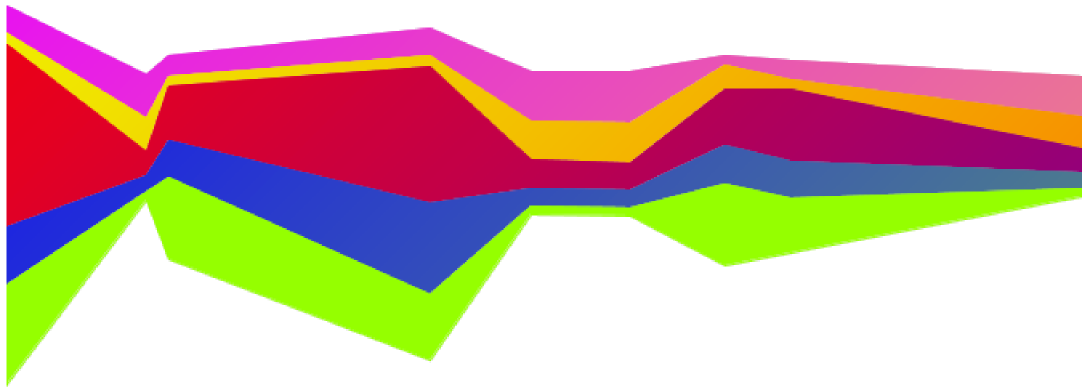
Most of the image seems to be fairly accurate with the exception of the ending, which completely misses the love shared between Jack and Sarah. The spike near the end I found to be particularly interesting. We will now look at Sarah’s character. Because this is now acting on a single character and not all the characters combined, there will be fewer segments.



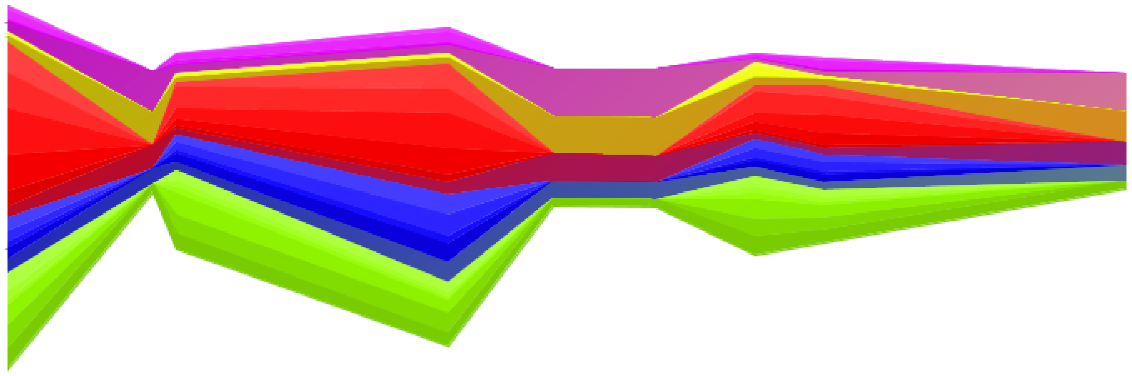
.



Similar to the other image this begins with a large anger and fear area. In fact the fear stays large right until the end, which is accurate as Sarah is constantly afraid of Jack’s judgment of her. Her anger is stronger near the beginning and subsides sooner than fear. This could be because once Jack arrives, she can no longer be angry but only sorry. Her love category seems to be a bit off. This is a common thread in our evaluations so far. Let’s look at Jack’s emotion image and see how it compares.



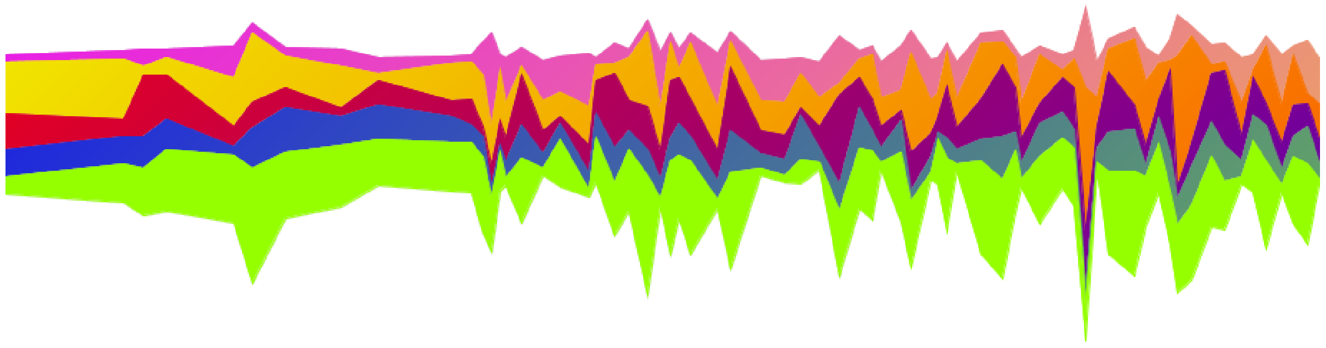
.

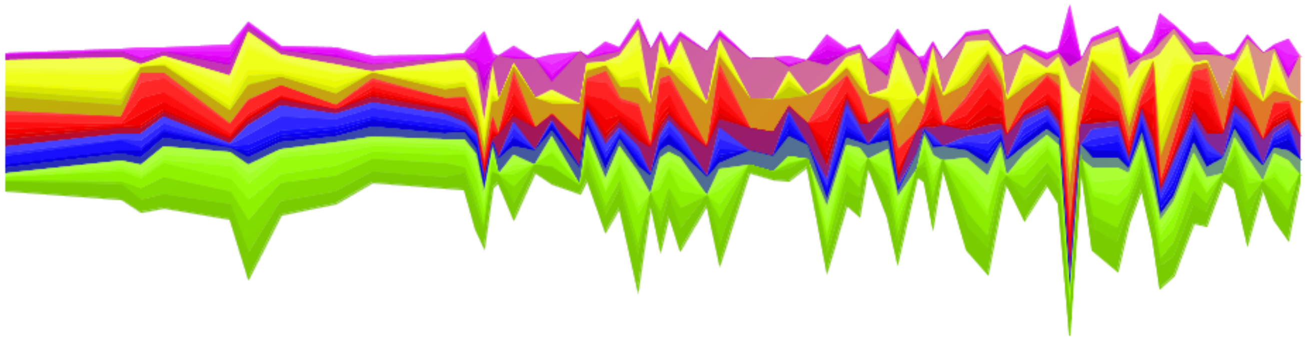


Jack's entire spectrum is shorter (blown up here); this could be because he is in less of the story than Sarah. His emotions seem to fluctuate greatly, with him being angry, sad, and fearful intermittently with the same general love and joy hovering at a low value for the majority of the image. The two spikes of these three emotions could be when he is describing how he looked for her at work when she wasn't there and then how his mother passed. These feelings all seem to subside in the end.

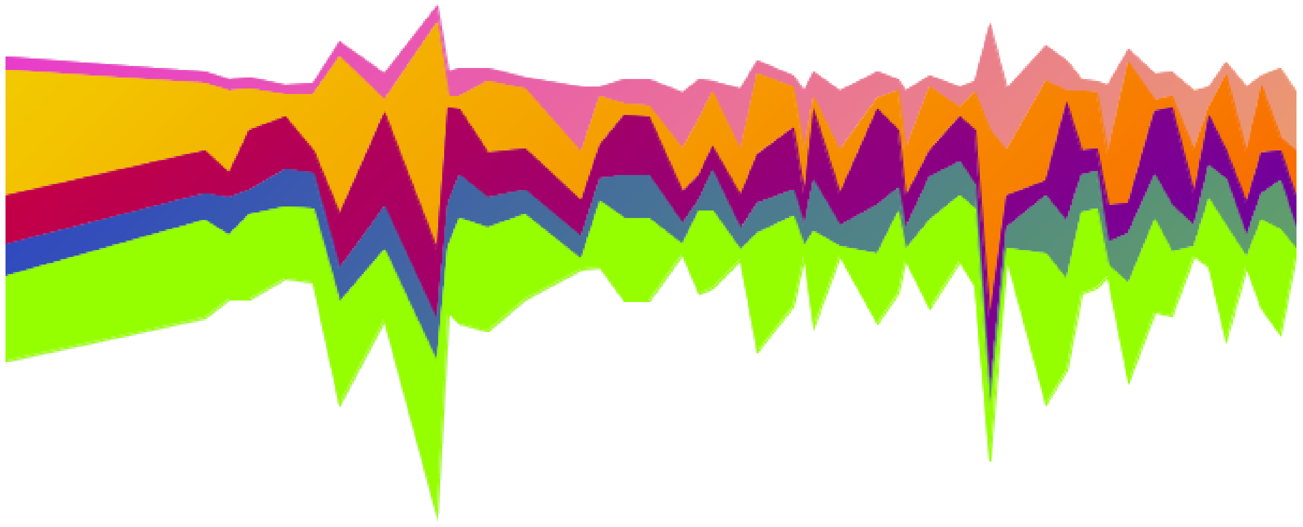
6.2 Text Two - Love is Better the Second Time Around

Seeing as our last text turned out to be much less loving than I'd thought it would be, I searched for a short 'romance' story. This is a surprisingly diverse field ranging from cute child crushes to the most imaginative pornography. I found a story that consisted of a few loving sex scenes and used the first chunk of it to visualize here. We will look at the story as a whole and then the main character, Kat. This is about a woman named Kat who meets up with an old flame after a horrible day and they begin to have sex about a third of the way through this sampled chunk.

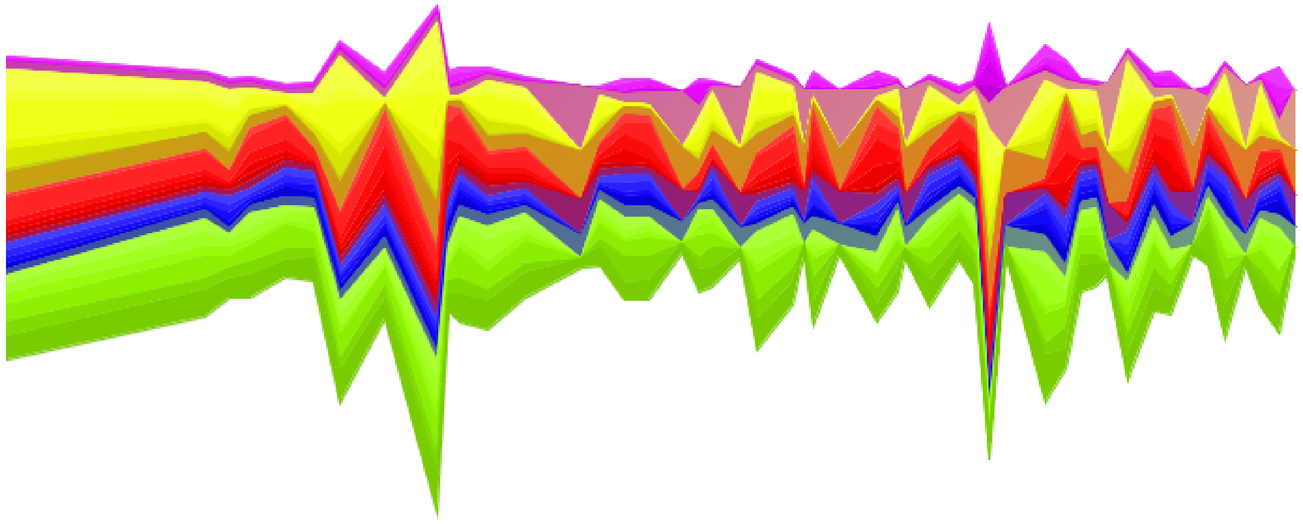




The first thing we notice about this story is the number of smaller spikes. This story is a bit longer than the last (though still not very long). We can see the beginning is pretty tepid in terms of emotions staying constant. Love seems to be a large section at the beginning. In this part of the story Kat is talking about how horrible her day has been, that she burnt her hand making coffee etc. The love sections seems to be a bit wrong in the beginning. Then anger spikes a bit, as does fear. Her car breaks down, which explains the growing fear and sadness categories as well as the tiny joy category. She does speak highly of others in this section, possibly accounting for the love section being large. The story then goes into a very fickle series of spikes to the end. One thing I do notice in this emotional jumble is that where love spikes, anger seems to subside, and fear bounces up and down consistently. This could be due to the fact that she keeps thinking and saying that she shouldn't have sex with her ex, and seems apprehensive. She cycles through the thoughts of enjoying herself and then fearing that she is not doing herself any favors by bringing out old feelings. Let's take a look at just Kat's emotion image.



.



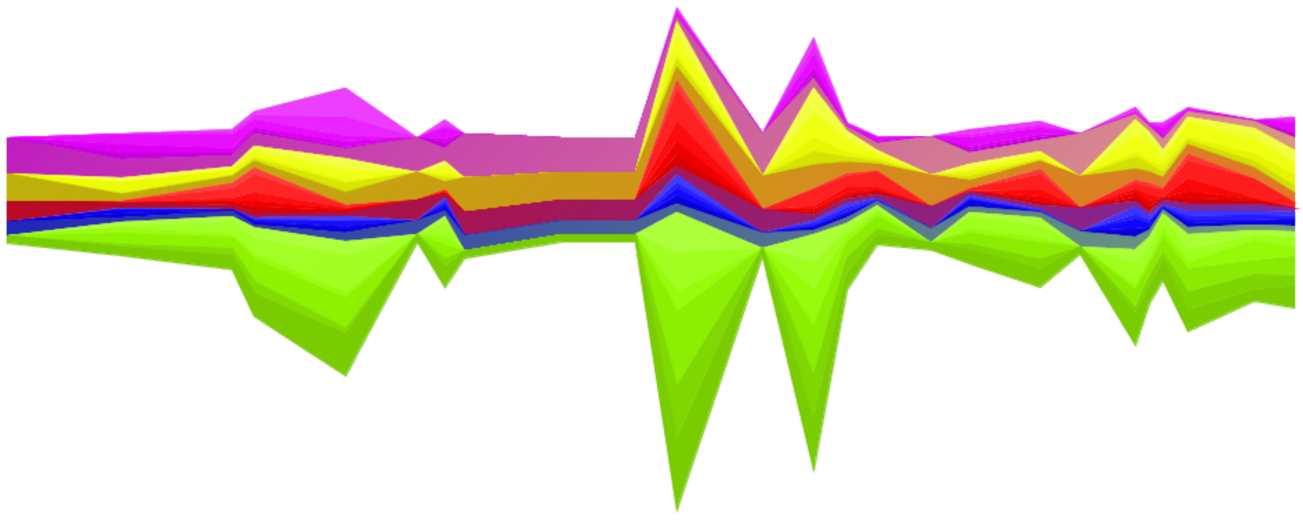
Kat’s image greatly resembles that of the entire story, probably due to the fact that this is a first person narrative and she is the ‘I’ voice. Due to this similarity, our visual analysis is essentially the same as the entire story.

6.3 How Successful is This?

In terms of the image aesthetics we are satisfied with the outcome. A user can define colors and use different variables to control how many emotions are on screen at each segment, how tall they can be with the threshold variable, and if secondary emotions are shown or not. The variation in the image that can be produced by changing 3 different variables is something we take pride it.

In terms of accuracy, we feel like we got about halfway to where we'd want to be. The method we chose in order to gather information is limited by the fact that it decontextualizes the words and then tries to give them meaning. After they are given meaning, the words are put back together in an attempt to recontextualize them. In retrospect, this approach seems a bit backwards and, if given the chance to start again, we'd probably try to gather more information than just a word for each Moment object. This is discussed further in the future work section below.

Something unforeseen was the length of the text being very important to the size of the image. If we want a TV proportioned image the text cannot be very long in order for use to have a clear visual of what is happening. In order to get a clear image of a large text, but still display it in a small space, we would have to be more discerning in what we displayed and somehow make the Segments to do more than just average Moments. Though we cannot truly get a compact visual on a novel right now, short stories are doable and can look quite nice when one switches the points, segment size, etc. variables to be ideal. One could make the image length a lot longer which would allow the longer stories to be presented, but a story 200 pages long would likely take up 30 feet lengthwise. Some might say this is a cool effect, but we really wanted an image that was more condensed. Maybe it isn't possible to condense a novel's worth of emotion into a meaningful 16:9 image. Right now if someone wants to see a novel they like, they could perform this analysis on a favorite chapter. For instance, if you liked chapter seven of Jane Austin's Pride and Prejudice(19), you'll love this next image.



When everything works, the image can be informative and engrossing, but when the data is too inaccurate, or there is too much data, the image can become confusing. We believe that for a first attempt, this project has merit. We hope the work we have done here will inspire others in the same way Martin Wattenberg inspired this project. We enter this into the visualization field as a stepping stone for ourselves and others to build upon conceptually or otherwise.

7. Future Work

This project has been a learning experience all throughout. If I could start over from September 2012 with all my retrospective knowledge I would have done things a bit differently, mostly in regards to the code structure. Looking forward, there aren't too many changes I could make without simply doing this process differently, but here are a few things I'd like to work on and improve. If anyone were to look at this project and want to expand on it, this is the section to read carefully.

With regards to extracting the mood/emotion of the novel, it seems like one can only go so far with contextless words gathered together. The first step to improve this aspect of the project would be to somehow contextualize the words. There might be other third party libraries that can do something like this; I imagine the code that makes google's search engine so robust these days is pretty good at contextualizing. In addition, we could try to pull in other words that the parser picks up. For example, adverbs that accent verbs could be counted together into one moment. We could have a weighted system that keeps track of the previous most relevant emotions and gives them a greater weight on the next few moments. This emotion momentum might help solve the issue of context, or it could help obscure the truth even more. Regardless it would be an avenue to explore. Also, semantic word relation is not perfect. If it makes a misjudgment once every five words, that is a 20% random element that will essentially smooth out the image with incorrect data.

When the program works well, our created image is something I am proud of. It is pretty clear what is happening once it is laid out for the viewer but it is also a neat looking image that could be a stand alone piece of art. The colors are user chosen, the segment size can drastically change the image, as can the start points variable. I believe this to be a good beginning to something that can be expanded on by myself and others. The code is currently a little messy and cleaning it up would be one of the first things to improve upon. Conceptually, code-wise, or otherwise I am open to suggestions from anyone (clintfmullins@gmail.com) and the entire project is available on my github (https://github.com/ClintFMullins).

8. Thanks

I'd like to thank everyone who helped with this project. Most of all Professor Bridget Baird who pushed me at every step, from deciding to do a thesis, presenting at a conference, to the last bit of execution, and yet she still let me have complete creative freedom over the project. I couldn't have asked for better advising. I'd like to thank my two other readers, Professor Christine Chung, who taught me data structures and game theory with expertise and warm encouragement and Charles Hartman, who gave us great advice in the project's early stages. Thanks to Professor Ozgur Izmirli and Professor Gary Parker for leading the research seminars these past two semesters, giving frank and constructive feedback throughout. To my family and friends, who listened intently and then forgot completely about the project many times over the last year, thank you for preparing me for the conference atmosphere and helping me to nail down exactly what I was doing. Thanks to Connecticut College for injecting the CS seeds in my brain like a virus; I don't think I'll ever be cured. And if you are none of the above, thanks for reading.

Best,
Clint

9. Works Cited

1. http://www.bewitched.com/
Mark Wattenberg's website with all his visualizations.

2. http://wordnet.princeton.edu/
WordNet is a graph of millions of words connected through their meaning similarities.

3. http://lyle.smu.edu/~tspell/jaws/
JAWS. A word semantic relation program for Java.

4. https://code.google.com/p/ws4j/
WS4J is another program similar to JAWS. A java word semantic relation program the runs on WordNet.

5. http://www.linguatools.de/disco/disco_en.html
DISCO, our word semantic program of choice. Similar to JAWS and WS4J but it works on different corpi.

6. http://www.natcorp.ox.ac.uk/corpus/index.xml
The British National Corpus website. It gives the details on the corpus and how it came to be.

7. http://nlp.stanford.edu/pubs/LREC06_dependencies.pdf
Stanford Parser Paper. Details of how it works.

8.http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/parser/lexparser/package-summary.html
Stanford Parser API for those of you who want in on the secrets.

9. http://nlp.stanford.edu/software/dependencies_manual.pdf
Stanford Parser Manual. This contains all the meanings for the tags the parser gives.

10. http://mallet.cs.umass.edu/topics.php
Mallet Software for Topic Modeling.

11. http://visual.ly/
Great examples of visualizations.

12. http://selection.datavisualization.ch/
More great examples of visualizations

13. http://www.webdesignerdepot.com/2009/06/50-great-examples-of-data-visualization/
You've had enough examples of visualizations, right? No? Alright, here is some more.

14. http://gunning-fog-index.com/
Gunning FOG index calculator

15. http://en.wikipedia.org/wiki/Gunning_fog_index
Gunning FOG index explanation with formula.

16. http://www.loverofdarkness.net/stories/story/45/t/kiss/o/0
Cheery site where Laurel: The Girl I Loved text is from. It is the example text I used all throughout the visual section.

17.http://www.angelfire.com/ga3/delayne/sstry/eternally.html
Comparison 1 short story “Eternally”

18. http://lovestories.hopcott.net/linnbelle/love-is-better/index.html
Comparison 2 short story “Love is Better the Second Time Around”

19. http://www.gutenberg.org/cache/epub/1342/pg1342.txt

Gutenberg.org offers many free books as text files. This is where we got Jane Austin's Pride and Prejudice.

10. Appendix of Code

Character.java

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collection;
import java.util.LinkedList;
import java.util.concurrent.ConcurrentHashMap;

public class Character {
	//private String[] fullName; //all names, for example: {"Mr.", "Franklin", "Jones"}
	private ConcurrentHashMap<String,Word> diction;
	private LinkedList occurences;  //holds all character mentions by index
	private String name;
	private int maleCount, femaleCount, gender;  //gender: 0 - female / 1 - male
	private ArrayList<Moment> emotionalRollercoaster; //keeps arrays of each emotion

	/**
	 * 1. Retrieves the character's name
	 * 2. Sets up the dictionary to save all character diction
	 * 3. Tries to define gender of character first by honorifics, then by name, if this fails
	 * we use a constant gender pronoun count to guess gender.
	 * 4. Sets up occurences which keeps track of all mentions of this character
	 * @param name
	 */
	public Character(String name, String fullName){
		System.out.print("New Character: "+name+" ||Fullname: "+fullName);
		this.name = name;
		emotionalRollercoaster = new ArrayList<Moment>();
		boolean assigned = false;
		if (!name.equals(fullName)){ //if there is a Mister, or Ms or what have you!
			boolean dot = fullName.contains(".")?true:false;
			System.out.print(" //Gender: ");
			if ("mr".contains(fullName.substring(0,(fullName.indexOf(" ")+(dot?-1:0)))))
{ //checks for Mr
				System.out.print("male");
				gender = 1;
				assigned = true;
			}
			else if ("msmrsmisses".contains(fullName.substring(0,fullName.indexOf(" ")+
(dot?-1:0)))){ //checks for ms, mrs, or misses
				System.out.print("female");
				gender = 0;
				assigned = true;
			}
			else{
```

```
                        System.out.print("unassigned1");
                    }
                }
                if (!assigned){
                    gender = Parse.nameGender(name);
                    if (gender==-1){
                        System.out.print("Unassigned2");
                    }
                    else if (gender==1){
                        System.out.print("male");
                        assigned = true;
                    }
                    else{
                        System.out.print("female");
                        assigned=true;
                    }
                }
                maleCount = femaleCount = 0;
                occurences = new LinkedList();
                diction = new ConcurrentHashMap<String,Word>();
            }

            /**
             * Writes out character moments to a specific character file [name].txt
             */
            public void writeOutMoments(boolean append){
                String momentString = "";
                for (Moment tempMoment: emotionalRollercoaster){
                    momentString+="!"+tempMoment.getWord()+"\n"+tempMoment.getX()+"\n";
                    for (MomentEmotion tempEmote: tempMoment.getMomentSpectrum()){
                        momentString+="+"+tempEmote.getPrimaryString()
+"\n"+tempEmote.getPrimaryVal()+"\n"; //add primary name and val (these are sorted by val so we
need the name)
                        for (MomentEmoteSec tempSecVal: tempEmote.getSecEmote()){
                            momentString+=tempSecVal.getSecondaryVal()+"\n";
                        }
                    }
                }
                try {
                  PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter("/VisualData/Characters/"+name, false)));
                  out.println(momentString);
                  out.close();
                } catch (Exception e) {
                  System.out.println("FAILED TO WRITE CHARACTER: "+name);
                }
            }

            /**
```

```
 * Adds a new word or adds occurrence of an old word that relates this character
 * @param newWord
 */
public void addWord(String newWord, int wordInd, String pOS){
	System.out.println("Name|Word|Index  "+name+"|"+newWord+"|"+wordInd);
	if (!diction.containsKey(newWord)){ //Word is not in list
		diction.put(newWord, new Word(newWord, wordInd, pOS)); //add word to list
	}
	else{
		diction.get(newWord).addOccurence(wordInd);
	}
	//here we add the moment to the emotional rollercoaster which holds all moments
	Moment newMoment = new Moment(diction.get(newWord));
	emotionalRollercoaster.add(newMoment);
	//if (name.equals("luke")){
		Visualize.addToAllMoments(newMoment);
	//}
}


/**
 * adds occurrence of this character
 * @param index
 */
public void addOccurence(int index){
	occurences.add(index);
}

/**
 * If there have been any gender assignment, returns true
 * @return
 */
public boolean isGender(){
	if (gender!=-1 || maleCount+femaleCount>0){
		return true;
	}
	return false;
}

/**
 * Get name of character
 * @return
 */
public String getName(){
	return name;
}

/**
 * adds weight to this as a male character
```

```
	 */
	public void maleMention(){
		maleCount++;
	}

	/**
	 * adds weight to this as a female character
	 */
	public void femaleMention(){
		femaleCount++;
	}

	/**
	 * Returns the gender assignment. If no assignment has been made then:
	 * Returns true (male) if the instances of male pronouns said directly after outweighs the female
pronouns
	 * @return
	 */
	public boolean isMale(){
		return (gender<0?(maleCount-femaleCount>0?true:false):(gender==0?false:true));
	}

	/**
	 * Returns a collection of words
	 * @return
	 */
	public Collection<Word> getDiction(){
		return diction.values();
	}

	public ArrayList<Moment> getEmotionalRollercoaster(){
		return emotionalRollercoaster;
	}
}
```

EmotionSpectrum.java

```
import de.linguatools.disco.DISCO;
import de.linguatools.disco.ReturnDataBN;
import de.linguatools.disco.ReturnDataCol;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.concurrent.ConcurrentHashMap;

public class EmotionSpectrum {
	/**
	 * Initializes the DISCO word association functions
	 */
	private static DISCO disco;
	private static PrimaryEmotion[] spectrum;
	private static int imageHeight;
	private static boolean influence; //variable that reflects whether secondary emotion relatedness
effect primary
	private static String[] primary = {"joy","love","anger","sadness","fear"};
	private static String[][] secondary =
			{{"cheerful", "bright", "content", "beauty", "optimism","relief"}, //joy
			 {"affection","lust","longing","attraction","compassion","infatuation","passion"}, //love
			 {"irritation", "kill", "rage", "disgust", "envy" ,"torment"}, //anger
			 {"cry", "tear", "shame", "neglect", "empty", "numb"}, //sadness
			 {"horror", "nervous","scare","dead","creep","sick"}}; //fear

//	private static String[] primary = {"female","male"};
//	private static String[][] secondary =
//			{
//			 {"compassion", "love", "soft", "beauty","giving"},
//			 {"strong", "tough", "hard", "cold","determined"}};

	/**
	 * 1. Sets up the emotional spectrum to be constantly referenced.
	 * This is where you change the primary, secondary emotions, weights, rgbs, height.
	 * Pretty much every important visual and technical definition. This is all accessible statically.
	 * 2. This must be initialized basically first in the program. It will error otherwise.
	 */
   public EmotionSpectrum(int imageHeight, int marginTop){
	String emotionalSpectrumWO= "";
	this.imageHeight = imageHeight;
	emotionalSpectrumWO+=imageHeight+"\n";
	influence = true;
       String discoDir = System.getProperty("user.dir")+"/PackageJars/DISCO/en-BNC-20080721";
       try {
				disco = new DISCO(discoDir, false);
			} catch (IOException e) {
				e.printStackTrace();
			}
       spectrum = new PrimaryEmotion[primary.length];
	double[][] weights = {{1,1,1,1,1,1,1},
					       {1,1,1,1,1,1,1},
					       {1,1,1,1,1,1,1},
					       {1,1,1,1,1,1,1},
					       {1,1,1,1,1,1,1}};
	int[] hsvs ={294, //love -- purple
				64, //Joy -- yellow
				0, //anger -- red
				242, //sadness -- blue
```

```
                                        86}; //fear -- green
        //here we split up the given height and give each secondary emotion a slot within our slot.
        int imageEmotionSlot = imageHeight/primary.length;
        int imageBottom = imageEmotionSlot+marginTop;
        int imageTop = 0+marginTop;
        for (int i=0;i<primary.length;i++){
                spectrum[i] = new PrimaryEmotion(primary[i], secondary[i], weights[i],
hsvs[i],imageTop, imageBottom);
                imageBottom=(i==primary.length-1?imageHeight:imageBottom+imageEmotionSlot);
                imageTop+=imageEmotionSlot;

                //writeout file appending
                emotionalSpectrumWO+=primary[i]+"\n"+  //add primary name
                                hsvs[i]+"|"+100+"|"+100+"\n"+ //primary color
                                                        imageTop+"|"+imageBottom+"\n"; //primary
spacing
                for (SecondaryEmotion curSec: getPrimary(primary[i]).getSecondary()){
                        int[] curHsv = curSec.getHsv();
                        emotionalSpectrumWO+=curSec.getSecondary()+"\n"+ //secondary name
                                        curHsv[0]+"|"+curHsv[1]+"|"+curHsv[2]+"|"+"\n"+ //secondary
spacing
                                        curSec.getyTop()+"|"+curSec.getyBottom()+"\n";  //secondary color

                }
        }
        writeOut(emotionalSpectrumWO);
  }

  /**
   * Writes out text
   * @param text
   */
        public void writeOut(String text){
                try {
                   PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter("VisualData/emotionalSpectrum", false)));
                   out.println(text);
                   out.close();
                } catch (Exception e) {
                   System.out.println();
                }
        }

  /**
   * 1. Given a string word, this will return an ArrayList of MomentEmotion
   * representation of that word's relation to that specific emotion.\
   * 2. boolean influence comes in here. If true, secondary emotions
   * add emotion value to their primary emotion. Otherwise, they don't. Should be true.
   * @param word
```

```
     * @throws IOException
     */
    public static MomentEmotion[] momentValue(String word){
        double primaryVal;
        double[] secondaryVal;
        MomentEmotion[] tempSpectrum = new MomentEmotion[primary.length];  //to be returned full
of emotions
        for (int i=0;i<primary.length;i++){
            try {
                PrimaryEmotion curEmote = getPrimary(primary[i]);
                primaryVal = disco.secondOrderSimilarity(word, curEmote.getPrimary());
                primaryVal = (primaryVal<0?0:primaryVal); //make sure the value isn't negative,
if it is, it is now 0
                secondaryVal = new double[curEmote.getSecondary().length];
                SecondaryEmotion[] tempSecEmotes = curEmote.getSecondary();
                for (int j=0;j<secondaryVal.length;j++){  //for each secondary emotion
                    secondaryVal[j] = disco.secondOrderSimilarity(word,
tempSecEmotes[j].getSecondary())*tempSecEmotes[j].getWeight(); //get relatedness value * weight of
sec. word
                    secondaryVal[j] = (secondaryVal[j]<0?0:secondaryVal[j]); //make sure the
value isn't negative, if it is, it is now 0
                    if (influence){
                        primaryVal+= (secondaryVal[j]/secondaryVal.length);  //secondary
value gets added top primary/number of secondary vals
                    }
                }
                primaryVal/=2; //this changes our value from 0-2 range to 0-1
                System.out.println("PrimaryVal: "+primaryVal);
                //moment emotion is created and added to our ArrayList to be returned
                tempSpectrum[i] = new
MomentEmotion(curEmote.getPrimary(),secondary[i],primaryVal,secondaryVal);
            }
            catch (Exception e){
                System.out.println("Word Relation Failure");
            }
        }
        return tempSpectrum;
    }

    public static PrimaryEmotion getPrimary(String emote){
        for (PrimaryEmotion primEmote: spectrum){
            if (primEmote.getPrimary().equals(emote)){
                return primEmote;
            }
        }
        System.out.println("No such thang.");
        return null;
    }
    public static PrimaryEmotion getPrimaryEmotion(String prime){
        for (PrimaryEmotion primTemp: spectrum){
            if (primTemp.getPrimary().equals(prime)){
                return primTemp;
            }
        }
        return null;
    }

    public static PrimaryEmotion[] getSpectrum(){
        return spectrum;
    }

    public static String[] getPrimaryStrings(){
        return primary;
    }

    public static String[][] getSecondaryStrings(){
        return secondary;
    }

}
```

Moment.java

```
import java.util.ArrayList;
import java.util.Collections;


public class Moment {
        private String word;  //word the moment surrounds
        private int x; //visual placement
        private MomentEmotion[] momentSpectrum;

        /**
         * A moment is defined by one word related to a Character and how that relates to our emotional
spectrum
         *
         * 1. String word is saved
         * 2. momentSpectrum is filled with MomentEmotion objects for relatedness
         * 3. Saved x position as the Word's index
         *
         * @param wordObj
         */
        public Moment(Word wordObj){
                word = wordObj.getWord();
                momentSpectrum = EmotionSpectrum.momentValue(this.word);
                x = wordObj.getRecentIndex();
                sortMomentSpectrum();  //sorts the momentEmotions by relevance
        }
```

```
	//Insertion sort used due to small list size (it should generally be less than 10).
	public void sortMomentSpectrum(){
		if (momentSpectrum.length>1){
			for (int i=1;i<momentSpectrum.length;i++){
				for (int j=i;j>0;j--){
					MomentEmotion tempEmote = momentSpectrum[j];
					if (tempEmote.getPrimaryVal()>momentSpectrum[j-
1].getPrimaryVal()){
						momentSpectrum[j] = momentSpectrum[j-1];
						momentSpectrum[j-1] = tempEmote;
					}
				}
			}
		}
	}

	public void printSpectrumOrderVals(){
		for (MomentEmotion tempMom: momentSpectrum){
			System.out.println(tempMom.getPrimaryString()+":
"+tempMom.getPrimaryVal());
		}
	}

	public MomentEmotion[] getMomentSpectrum(){
		return momentSpectrum;
	}
	public String getWord(){
		return word;
	}
	public int getX(){
		return x;
	}
}
```

MomentEmoteSec.java

```
public class MomentEmoteSec {
	private String secondaryString;
	private int size,y;
	private double secondaryVal;

	public MomentEmoteSec (String secondaryString, double secondaryVal){
		this.secondaryString = secondaryString;
		this.secondaryVal = secondaryVal;
	}

	public String getSecondaryString() {
		return secondaryString;
	}

	public void setSecondaryString(String secondaryString) {
		this.secondaryString = secondaryString;
	}

	public double getSecondaryVal() {
		return secondaryVal;
	}

	public void setSecondaryVal(double secondaryVal) {
		this.secondaryVal = secondaryVal;
	}

	public int getSize() {
		return size;
	}

	public void setSize(int size) {
		this.size = size;
	}

	public int getY() {
		return y;
	}

	public void setY(int y) {
		this.y = y;
	}
}
```

MomentEmoteSpace.java

```
public class MomentEmoteSpace {
	private String emotion;
	private boolean primary;
	private int top, bottom, points;
	private double value;
	private MomentEmoteSpace[] secondary;

	/**
	 * Primary Emote space initializer
	 * @param prime
	 */
	public MomentEmoteSpace(MomentEmotion momEmote){
		emotion = momEmote.getPrimaryString();
		primary = true;
		top = EmotionSpectrum.getPrimaryEmotion(emotion).getyTop();
		bottom = EmotionSpectrum.getPrimaryEmotion(emotion).getyBottom();
		secondary = new MomentEmoteSpace[momEmote.getSecEmote().length];
//secondary emotion list created
		for (int i=0; i<momEmote.getSecEmote().length;i++){
			String seconString =
EmotionSpectrum.getPrimaryEmotion(emotion).getSecondary()[i].getSecondary();
			secondary[i] = new MomentEmoteSpace(seconString,
momEmote.getSecondaryVal(seconString));
		}
	}
	/**
	 * If this is a secondary, we initialize here. No bounds are set yet. That
comes later.
	 * @param emotion
	 * @param value
	 */
	public MomentEmoteSpace(String emotion, double value){
		this.emotion = emotion;
		primary = false;
		this.value = value;
	}

	/**
	 * secondary Emote space initializer
	 * @param sec
	 */
	public MomentEmoteSpace(SecondaryEmotion sec){
		emotion = sec.getSecondary();
		primary = false;
		top = sec.getyTop();
		bottom = sec.getyBottom();
	}

	public void assignSecondaries(){
		if (primary){ //can't assign secondaries if you are a secondary!
			int yspace = bottom - top;
			double total = 0;
			for (MomentEmoteSpace sec: secondary){ //get the total value
				total+=sec.getValue();
			}
			System.out.println("SECONDARYSPACING!\n"+top + " " + bottom);
			for (int i=0;i<secondary.length;i++){
				double percent = secondary[i].getValue()/total; //get our
percentage of the total
				int curYSpace = (int)(yspace*percent); //get our actual y-
space value
				if (i==0){ //if this is the first category, then the y-top
is the y-top of the primary
					secondary[i].setTop(top);
				}
				else{
					secondary[i].setTop(secondary[i-1].getBottom());
				}
				if (i==secondary.length-1){
					secondary[i].setBottom(bottom);
				}
				else{
					secondary[i].setBottom(secondary[i].getTop()
+curYSpace);
				}
				System.out.println(secondary[i].getEmotion()+ " "+
secondary[i].getTop()+ " " + secondary[i].getBottom());
			}

		}
	}

	public MomentEmoteSpace[] getSecondary(){
		if (primary){
			return secondary;
		}
		return null;
	}

	public double getValue(){
		return value;
	}


	public boolean isPrimary() {
		return primary;
	}

	public int getTop() {
		return top;
	}

	public void setTop(int top) {
		this.top = top;
	}

	public int getBottom() {
		return bottom;
	}

	public void setBottom(int bottom) {
		this.bottom = bottom;
	}

	public String getEmotion() {
		return emotion;
	}

	public void setEmotion(String emotion) {
		this.emotion = emotion;
	}

	public int getPoints() {
		return points;
	}

	public void setPoints(int points) {
		this.points = points;
	}
}
```

MomentEmotion.java

```
import java.util.Comparator;


/**
 * @author Clint Mullins
 *
 */
public class MomentEmotion{
	private String primaryString; //name of primary emotion
	private double primaryVal;	//primary emotion relatedness
	private MomentEmoteSec[] secEmote; //holds secondary emotions with name, 
value, and size

	/**
	 * One primary emotion with its secondary emotions defined in one Moment. 
This saves the relatedness.
	 * 
	 * @param primaryString
	 * @param secondaryString
	 * @param primary
	 * @param secondary
	 */
	public MomentEmotion(String primaryString, String[] secondaryString, double 
primary, double[] secondaryVal){
		this.primaryString = primaryString;
		this.primaryVal = primary;
		secEmote = new MomentEmoteSec[secondaryString.length];
		for (int i=0;i<secondaryString.length;i++){
			secEmote[i]=new 
MomentEmoteSec(secondaryString[i],secondaryVal[i]);
		}
		sortSecondary();
	}

	/**
	 * Sorts the secondary emotions
	 */
	public void sortSecondary(){
		for (int i=1;i<secEmote.length;i++){
			for (int j=i;j>0;j--){
				MomentEmoteSec tempEmoteSec = secEmote[j];
				if (tempEmoteSec.getSecondaryVal()>secEmote[j-
1].getSecondaryVal()){
					secEmote[j] = secEmote[j-1];
					secEmote[j-1] = tempEmoteSec;
				}
			}
		}
	}

	public double getSecondaryVal(String sec){
		for (MomentEmoteSec momSec: secEmote){
			if (momSec.getSecondaryString().equals(sec)){
				return momSec.getSecondaryVal();
			}
		}
		return 0.0;
	}

	public String getPrimaryString() {
		return primaryString;
	}

	public void setPrimaryString(String primaryString) {
		this.primaryString = primaryString;
	}

	public double getPrimaryVal() {
		return primaryVal;
	}

	public void setPrimaryVal(double primaryVal) {
		this.primaryVal = primaryVal;
	}

	public MomentEmoteSec[] getSecEmote() {
		return secEmote;
	}

	public void setSecEmote(MomentEmoteSec[] secEmote) {
		this.secEmote = secEmote;
	}
}
```

Novel.java

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.LinkedList;
import java.util.concurrent.ConcurrentHashMap;


/**
 * If characters have the same name, that is implied connection
 *
 * @author ClintFrank
 *
 */

public class Novel {
	//FOGVVV
	private static int letterCount, sentenceLength, wordCount, sentenceCount, speechCount,
sylCount, totalSylCount, compWordCount, index;
	private static double averageLetWord, averageSylWord, averageWordSen, FOG;
	private static boolean speechFlag, senEndFlag, preVowelFlag;
	private static String punct, stringChar;
	//FOG^^^
	private static String[] words, depList, tokList;
	private static int parseReaderIndex;
	private String cleanFile, tokenized, depended, title, author, curSentence, curWord;
	private boolean parseExists;
	private ConcurrentHashMap<String,Character> cast;
	private LinkedList<String> listOfCharacters;
	private Parse parser;
	private String curCharMale, curCharFemale, curCharUni; //keeps track of the last Character
mentioned, male, female, and before assignment
```

```
	public Novel (String title, String author, int imageHeight, int marginTop){
		this.title = title;
		this.author = author;
		listOfCharacters = new LinkedList<String>();
		@SuppressWarnings("unused")
		EmotionSpectrum emotionSpec = new EmotionSpectrum(imageHeight,marginTop);
//we just need to initialize. All methods are static.
		parser = new Parse();
		punct = ".?!";
		tokenized=depended="";
		cast = new ConcurrentHashMap<String,Character>();
		cleanFile = Parse.cleanFile(title);  //here we get a clean copy of the book
//		cleanFile = "Mr. Hannifan was a good man. " +
//					"He was always near Steph. " +
//					"Until one day Dr. Hannifan slapped her. " +
//					"Mike told him that she was upset." +
//					"Mike cried the whole day." +
//					"Mr. Hannifan then shouted, 'Go kill yourself!'";
		parseReaderIndex = 0; //keeps track of the sentence location in lists below
		String depPath = "VisualData/Parsed/depended"+title+cleanFile.charAt(0)+".txt";
		String tokPath = "VisualData/Parsed/tokenized"+title+cleanFile.charAt(1)+".txt";
		if (fileExists(depPath)&&fileExists(tokPath)){
			parseExists = true;
			depList = readFile(depPath).split("\n");
			tokList = readFile(tokPath).split("\n");
			characterExtraction();
		}
		else{
			parseExists = false;
			characterExtraction();
			writeOut(tokenized,tokPath);
			writeOut(depended,depPath);
		}
	}

	/**
	 * returns text length for visualization
	 * @return
	 */
	public int getTextLength(){
		return cleanFile.length();
	}

	public boolean fileExists(String path){
		if (new File(path).exists()){
			return true;
		}
		return false;
```

```
    }

    public String readFile(String path){
        try{
            String longString="";
            String curLine;
            BufferedReader br = new BufferedReader(new FileReader(path));
            while ((curLine = br.readLine()) != null) {
                longString+=curLine+"\n";
            }
            br.close();
            return longString;
        }
        catch(Exception e){
            System.out.println("FAILED TO READ");
            return null;
        }

    }

    public void characterExtraction(){
        curCharMale = curCharFemale = curCharUni = "0";
        curWord = curSentence = "";
        for (index=0;index<cleanFile.length();index++){
            stringChar = String.valueOf(cleanFile.charAt(index));
            //System.out.println(stringChar);
            //Counting the length of dialogs
            curWord+=stringChar;
            curSentence+=stringChar; //current sentence being created
            if (speechFlag){
                setSpeechCount(getSpeechCount() + 1);
            }
            //space
            if (" ".contains(stringChar)){
                //space after word (new word)
                if (senEndFlag==false){
                    //Complex word
                    if (sylCount>3){
                        compWordCount++;
                    }
                    if (cast.contains(curCharUni)){  //if the last reference proper noun
is a character
                        curWord = curWord.trim().toLowerCase(); //no whitespace
and lowercase
                        if ("sheher".contains(curWord)){ //if feminine pronoun
                            System.out.println("FEMASSIGN");
                            cast.get(curCharUni).femaleMention(); //adds to
female chance of last referenced char
                        }
                        else if ("hehim".contains(curWord)){
                            System.out.println("MALEASSIGN");
                            cast.get(curCharUni).maleMention(); //adds to the
male chance of last referenced char
                        }
                    }
                    setWordCount(getWordCount() + 1);
                    sylCount=0;
                    curWord="";
                    preVowelFlag=false;
                }
                //space after end of sentence
                else{
                    senEndFlag=false;
                }
            }
            //end of sentence (New Sentence)
            else if (punct.contains(stringChar)){
                try{
                    String check = curSentence.substring(curSentence.length()-
3,curSentence.length()); //created to check for Ms. Mr. Dr.
                    if (check.contains("Mr.")||check.contains("Ms.")||
check.contains("Dr.")){ //if Dr. Mr. Ms. then sentence isn't over
                        continue;
                    }
                }
                catch (Exception e){
                    System.out.println("MR MS DR CHECK OUT OF RANGE");
                }
                //System.out.println("[word] \n[senClose]");
                curSentence = curSentence.trim();  //gets rid of whitespace in the
beginning or end of a sentence
                System.out.println(curSentence);
                characterIdentify(curSentence); //sends sentence for character
identification
                characterPersonify(curSentence); //sends sentence for character
personification
                setSentenceCount(getSentenceCount() + 1);
                setWordCount(getWordCount() + 1);
                curSentence="";
                senEndFlag=true;
            }
            //dialog
            else if ("\"".contains(stringChar)){
                //open dialog
                if (speechFlag==false){
                    //System.out.print("[sOpen] ");
                    speechFlag=true;
                }
```

```
					//closed dialog
					else {
						//System.out.print("[sClose] ");
						setSpeechCount(getSpeechCount() + 1);
						speechFlag=false;
					}
				}
				//letters
				else{
					//Vowel
					if ("aeiouy".contains(stringChar)){
						//Isolated Vowel
						if (preVowelFlag==false){
							//System.out.print("[s]");
							sylCount++;
							totalSylCount++;
							preVowelFlag=true;
						}
					}
					//Consonant
					else{
						preVowelFlag=false;
					}
					setLetterCount(getLetterCount() + 1);
				}

			}
			averageWordSen = ((double) wordCount)/sentenceCount;
			setAverageLetWord(((double) letterCount)/wordCount);
			setAverageSylWord(((double) totalSylCount)/wordCount);
			setFOG((0.4)*(averageWordSen + 100*(compWordCount/wordCount)));
			characterWriteOut();
		}

		/**
		 * This class looks at all the characters and writes out their data to separate [name].txt files
within VisualData/Characters
		 */
		public void characterWriteOut(){
			for (String tempActor: listOfCharacters){  //here we cycle through all added names
				File file = new File("VisualData/Character/", cast.get(tempActor).getName() +
".txt"); //file is created for character
				cast.get(tempActor).writeOutMoments(false);
			}
		}

		public LinkedList<String> getListOfCharacters(){
			return listOfCharacters;
		}
```

```
	public Character getCharacter(String charName){
		return cast.get(charName);
	}

	public void writeOut(String text, String path){
		try {
		   PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter(path, false)));
		   out.println(text);
		   out.close();
		} catch (Exception e) {
		   System.out.println("FAILED TO WRITE: "+path);
		}
	}

	/**
	 * Identifies Proper nouns from texts and makes them characters
	 * @param sentence
	 */
	public boolean characterIdentify(String sentence){
		String senWithPOS;
		if (parseExists){
			 senWithPOS = tokList[parseReaderIndex];
		}
		else{
			senWithPOS = parser.getPOS(sentence);
			tokenized+=senWithPOS+"\n";
		}
		System.out.println("POS "+senWithPOS);
		int start, i;
		boolean charInSen = false;
		while (true){
			//System.out.println("loop1 STUCK?");
			if ((start = senWithPOS.indexOf("NNP")) == -1){
				return charInSen;
			}
			charInSen = true;
			start+=4;
			i = start;
			String name = "";
			while (true){
				while (!String.valueOf(senWithPOS.charAt(i)).equals(")")){ //grabs the
name
					name+=String.valueOf(senWithPOS.charAt(i));
					i++;
				}
				if (senWithPOS.length()>=i+7 &&
senWithPOS.substring(i+3,i+6).equals("NNP")){ //is there more to this name? ex. John Smith
					i+=7;
					name+=" ";
				}
				else{
					break;
				}
			}
			addCharacter(name.trim().toLowerCase()); //here we send the name to be added
			senWithPOS = senWithPOS.substring(i,senWithPOS.length());
		}
	}

	/**
	 *
	 * @param sentence
	 */
	public void characterPersonify(String sentence){
		String senWithDep;
		if (parseExists){
			senWithDep = depList[parseReaderIndex++];
		}
		else{
			senWithDep = parser.getDep(sentence);
			depended+=senWithDep+"\n";
		}
		boolean adv;
		String checkWithDep = "";
		System.out.println("Dep "+senWithDep);
		int start;
		String word, name, pOS, wordInd;
		pOS = "nsubj";
		while ((start = senWithDep.indexOf("nsubj"))!=-1){
			word = name = wordInd = "";
			if (senWithDep.contains("advmod")){
				checkWithDep = senWithDep;  //for advmod
				adv = true;
			}
			else{
				adv = false;
			}
			while (!String.valueOf(senWithDep.charAt(start)).equals("(")){ //here we get to
the word
				start++;
			}
			start++; //skips the "("
			while (!String.valueOf(senWithDep.charAt(start)).equals("-")){ //we record the
word
				word+=senWithDep.charAt(start);
				start++;
			}
```

```
				start++; //skips the "-"
				while (!String.valueOf(senWithDep.charAt(start)).equals(",")){  //we record the 
word index
					wordInd+=senWithDep.charAt(start);
					start++;
				}
				start+=2;  //skips the "-#, "
				while (!String.valueOf(senWithDep.charAt(start)).equals("-")){  //here we get the 
name
					name+=senWithDep.charAt(start);
					start++;
				}
				name = name.trim().toLowerCase();
				word = word.trim().toLowerCase();
				System.out.println("Name: "+name+" Verb: "+word);
				if (adv){
					String advWord = "";
					String checkWord = "";
					int curChar;
					checkWithDep = 
checkWithDep.substring(checkWithDep.indexOf("advmod"),checkWithDep.length());
					curChar
 = 0;
//					while (!String.valueOf(checkWithDep.charAt(curChar)).equals("(")){
//
//					}
					curChar++; //skips the "("
				}
				if (name.equals("he")){
					System.out.println("adding word - curCharMale: "+curCharMale);
					if (!curCharUni.equals("0")){
						cast.get(curCharMale.equals("0")?
curCharUni:curCharMale).addWord(word, getNum(wordInd)+index, pOS);
					}
				}
				else if (name.equals("she")){
					System.out.println("adding word - curCharFemale: "+curCharFemale);
					cast.get(curCharFemale.equals("0")?
curCharUni:curCharFemale).addWord(word, getNum(wordInd)+index, pOS);
				}
				else if (cast.containsKey(name)){
					curCharUni = name;  //sets last mentioned male character as current
					System.out.println("adding word");
					cast.get(name).addWord(word, getNum(wordInd)+index, pOS);
				}
				senWithDep = senWithDep.substring(start+1, senWithDep.length());
				//System.out.println(senWithDep);
			}
		}
	/**
	 * kills all non numbers left of actual numbers in String
	 * @param numString
	 * @return
	 */
	public int getNum(String numString){
		if (numString.contains("\"")){
			System.out.println("CONTAINS IT");
			numString = numString.substring(0,numString.length()-1);
		}
		while (true){
			try {
				return Integer.parseInt(numString);
			}
			catch(Exception e){
				return getNum(numString.substring(1,numString.length()));
			}
		}
	}

	public void addCharacter(String name){
		name = name.trim();
		String fullName = name;  //keeps full name in case we can assign gender here
		if (name.contains(" ")){  //if name has a Mr. Dr. etc we strip it
			name = name.substring(name.indexOf(" ")+1,name.length());
		}
		curCharUni = name;
		if (cast.containsKey(name)){
			System.out.print("Char Exists");
		}
		else{
			Character tempNewChar = new Character(name,fullName);
			cast.put(name, tempNewChar);  //character is added to hashmap
			listOfCharacters.add(name); //string name of character is added to this list for
easy cycling later
		}
		cast.get(name).addOccurence(index);  //says index is sentence index
		if (cast.get(name).isGender()){ //if there has been any gender assignment
			if (cast.get(name).isMale()){  //if the character is male
				curCharMale = name;			//this is the last male referenced
character
			}
			else{
				curCharFemale = name;	//otherwise it is the last female referenced
character
			}
		}
}

public void setCleanFile(String cleanFile){
	this.cleanFile = cleanFile;
}

public String getTitle(){
	return title;
}

public void setAuthor(String author){
	this.author = author;
}

public static int getLetterCount() {
	return letterCount;
}

public static void setLetterCount(int letterCount) {
	Novel.letterCount = letterCount;
}

public static int getSentenceLength() {
	return sentenceLength;
}

public static void setSentenceLength(int sentenceLength) {
	Novel.sentenceLength = sentenceLength;
}

public static int getWordCount() {
	return wordCount;
}

public static void setWordCount(int wordCount) {
	Novel.wordCount = wordCount;
}

public static int getSentenceCount() {
	return sentenceCount;
}

public static void setSentenceCount(int sentenceCount) {
	Novel.sentenceCount = sentenceCount;
}

public static double getAverageSylWord() {
	return averageSylWord;
}
```

```
	public static void setAverageSylWord(double averageSylWord) {
		Novel.averageSylWord = averageSylWord;
	}

	public static double getAverageLetWord() {
		return averageLetWord;
	}

	public static void setAverageLetWord(double averageLetWord) {
		Novel.averageLetWord = averageLetWord;
	}

	public static double getFOG() {
		return FOG;
	}

	public static void setFOG(double fOG) {
		FOG = fOG;
	}

	public static int getSpeechCount() {
		return speechCount;
	}

	public static void setSpeechCount(int speechCount) {
		Novel.speechCount = speechCount;
	}
}
```

Parse.java

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Collection;
import java.util.Scanner;

import cc.mallet.grmm.types.Tree;
import cc.mallet.types.FeatureConjunction.List;
import edu.stanford.nlp.parser.lexparser.LexicalizedParser;
import edu.stanford.nlp.trees.GrammaticalStructure;
import edu.stanford.nlp.trees.GrammaticalStructureFactory;
import edu.stanford.nlp.trees.PennTreebankLanguagePack;
import edu.stanford.nlp.trees.TreebankLanguagePack;
```

```
public class Parse {
	private LexicalizedParser lp;

	public Parse(){
		lp =
LexicalizedParser.loadModel("edu/stanford/nlp/models/lexparser/englishPCFG.ser.gz");
	}

	/**
	 *
	 * @param name
	 */
	public static int nameGender(String name){
		try{
			Scanner scan = new Scanner(new File("../referenceFiles/male_First.txt"));
			while (scan.hasNext()) {
			   if (scan.next().equals(name)){  //if the name is in the male database
				scan.close();
				return 1;
			   }
			}
			scan = new Scanner(new File("../referenceFiles/female_First.txt"));
			while (scan.hasNext()) {
			   if (scan.next().equals(name)){  //if the name is in the female database
				scan.close();
				return 0;
			   }
			}
			scan.close();
			return -1;
		}
		catch (Exception e){
			System.out.println("File Read Failed\n\n\n\n");
			return -1;
		}
	}

	/**
	 * Given ONE SENTENCE, this will return the parts of speech of all words within that
sentence.
	 * @param sentence (String) one sentence, no more no less.
	 * @return (String) that same sentence with POS tagged. Output looks like this:
	 *
	 * (ROOT (S (VP (VB Hey) (S (NP (EX there)) (NP (NNP John)))) (. .)))
	 */
	public String getPOS (String sentence){
		edu.stanford.nlp.trees.Tree parse = lp.apply(sentence);
		//System.out.println(parse.toString());
		return parse.toString();
```

```
        }

        /**
         * Given ONE SENTENCE, this will return the relationships (dependencies between words) of
all words within that sentence.
         * @param sentence
         * @return (String) given sentence with relationships tagged. Output looks like this:
         *
         * [root(ROOT-0, Hey-1), expl(John-3, there-2), nsubj(John-3, there-2), xcomp(Hey-1, John-3)]
         */
        public String getDep(String sentence){
           TreebankLanguagePack tlp = new PennTreebankLanguagePack();
           GrammaticalStructureFactory gsf = tlp.grammaticalStructureFactory();
           edu.stanford.nlp.trees.Tree parse = lp.apply(sentence);
                //System.out.println();
                GrammaticalStructure gs = gsf.newGrammaticalStructure(parse);
                @SuppressWarnings("rawtypes")
                Collection tdl = gs.typedDependenciesCCprocessed(true);
                //System.out.println(tdl.toString());
                return tdl.toString();
        }

   /**
    * Searches for a
    * @param fileName
    * @return
    */
   public static String cleanFile(String fileName) {
        System.out.println("File Cleaned");
        File novelFile = new File("books/"+fileName+".txt");
     StringBuilder contents = new StringBuilder();
     String line;
     try {
        BufferedReader input =  new BufferedReader(new FileReader(novelFile));
        try {
                boolean read = false;
                while (( line = input.readLine()) != null){
                        if (!read && line.contains("*** START OF THIS PROJECT GUTENBERG
EBOOK")){
                                contents = new StringBuilder();
                                read=true;
                        }
                        else if (read && line.contains("*** END OF THIS PROJECT GUTENBERG
EBOOK")){
                                break;
                        }
                        contents.append(line);
                        contents.append(System.getProperty("line.separator"));
                }
```

```
            }
            finally {
                    input.close();
            }
        }
        catch (IOException ex){
                    ex.printStackTrace();
        }
        try{
            // Create file
            FileWriter fstream = new FileWriter("books/"+fileName);
            BufferedWriter out = new BufferedWriter(fstream);
            out.write(contents.toString());
            //Close the output stream
            out.close();
        }catch (Exception e){//Catch exception if any
            System.err.println("Error: " + e.getMessage());
        }
        return contents.toString();
    }

    public static void main(String[] args){
            Parse test = new Parse();
            String sentence = "Monica is so disgustingly pretty";
            System.out.println(test.getDep(sentence));
            System.out.println(test.getPOS(sentence));
    }
}
```

PrimaryEmotion.java

```
import java.awt.Color;

public class PrimaryEmotion {
      private String primaryString;   //primary emotion for the class
      private SecondaryEmotion[] secondary;  //emotions that comprise/help define
the primary
      private float[] hsv = {0,100,100}; //color for this Emotion
      private Color primeColor;
      private int yTop, yBottom, yCenter; //represent graphically the top and
bottom of our image

      public PrimaryEmotion(String primaryString, String[] secondaryStrings,
double[] weights, int hue, int yTop, int yBottom){
            this.primaryString = primaryString;
            this.secondary = new SecondaryEmotion[secondaryStrings.length];
            hsv[0]=hue;
            this.yTop = yTop;
            this.yBottom = yBottom;
            this.setyCenter(Math.abs(yBottom-yTop)/2+yTop);
            int imageEmotionSlot = (yBottom-yTop)/secondaryStrings.length; //space
for each secondary
            System.out.println("imageEmotionSlot: "+imageEmotionSlot);
```

```
	int imageBottom = imageEmotionSlot+yTop;
	int imageTop = yTop;
	int svLow = 100-(secondaryStrings.length*4); //decides how low saturation and
brightness values can be for secondary color
	int[] secHsv = {hue,svLow,100}; //actual value for sat and bright for each
secondary color
	int svChange = ((100-svLow)*2)/(secondaryStrings.length-1); //gets our
differences
	System.out.println("PRIMARY "+primaryString+":");
		for (int i=0; i<secondaryStrings.length;i++){
			secondary[i] = new
SecondaryEmotion(secondaryStrings[i],weights[i],secHsv,imageTop,imageBottom);
			imageBottom=(secondaryStrings.length-1==i?
yBottom:imageBottom+imageEmotionSlot);
			imageTop+=imageEmotionSlot;
			//we want to run along the saturation and then the value for each
hue
			if (secHsv[1]<100){ //this places saturation of the given hue
that start from svLow and go to 100
				if (secHsv[1]+svChange>=100){
					secHsv[2]=100-(svChange-(100-secHsv[1]));  //carries
over the difference
					secHsv[1]=100;  //sets the saturation at 100
				}
				else{ //this starts at 100 and decreases brightness (v)
until there are no more secondary emotions
					secHsv[1]+=svChange;
				}
			}
			else{
				secHsv[2]-=svChange; //descend from 100 brightness(v) to
about svLow
			}
		}
		setPrimeColor(Color.getHSBColor(hsv[0]/360, hsv[1]/100, hsv[2]/100));
	}

	public SecondaryEmotion getSecondaryEmotion(String emote){
		for (SecondaryEmotion tempSec: secondary){
			if (tempSec.getSecondary().equals(emote)){
				return tempSec;
			}
		}
		return null;
	}

	public String getPrimary() {
		return primaryString;
	}

	public SecondaryEmotion[] getSecondary() {
		return secondary;
	}

	public float[] getHSV() {
		return hsv;
	}

	public int getyTop() {
		return yTop;
	}

	public int getyBottom() {
		return yBottom;
	}

	public int getyCenter() {
		return yCenter;
	}

	public void setyCenter(int yCenter) {
		this.yCenter = yCenter;
	}

	public Color getPrimeColor() {
		return primeColor;
	}

	public void setPrimeColor(Color primeColor) {
		this.primeColor = primeColor;
	}
}
```

SecondaryEmotion.java

```
import java.awt.Color;

public class PrimaryEmotion {
	private String primaryString;   //primary emotion for the class
	private SecondaryEmotion[] secondary;  //emotions that comprise/help define
the primary
	private float[] hsv = {0,100,100}; //color for this Emotion
	private Color primeColor;
	private int yTop, yBottom, yCenter; //represent graphically the top and
bottom of our image

	public PrimaryEmotion(String primaryString, String[] secondaryStrings,
double[] weights, int hue, int yTop, int yBottom){
		this.primaryString = primaryString;
		this.secondary = new SecondaryEmotion[secondaryStrings.length];
		hsv[0]=hue;
		this.yTop = yTop;
		this.yBottom = yBottom;
		this.setyCenter(Math.abs(yBottom-yTop)/2+yTop);
		int imageEmotionSlot = (yBottom-yTop)/secondaryStrings.length; //space
for each secondary
		System.out.println("imageEmotionSlot: "+imageEmotionSlot);
	int imageBottom = imageEmotionSlot+yTop;
	int imageTop = yTop;
	int svLow = 100-(secondaryStrings.length*4); //decides how low saturation and
brightness values can be for secondary color
	int[] secHsv = {hue,svLow,100}; //actual value for sat and bright for each
secondary color
	int svChange = ((100-svLow)*2)/(secondaryStrings.length-1); //gets our
differences
	System.out.println("PRIMARY "+primaryString+":");
		for (int i=0; i<secondaryStrings.length;i++){
			secondary[i] = new
SecondaryEmotion(secondaryStrings[i],weights[i],secHsv,imageTop,imageBottom);
                  imageBottom=(secondaryStrings.length-1==i?
yBottom:imageBottom+imageEmotionSlot);
                  imageTop+=imageEmotionSlot;
                  //we want to run along the saturation and then the value for each
hue
                  if (secHsv[1]<100){ //this places saturation of the given hue
that start from svLow and go to 100
                        if (secHsv[1]+svChange>=100){
                              secHsv[2]=100-(svChange-(100-secHsv[1]));  //carries
over the difference
                              secHsv[1]=100;  //sets the saturation at 100
                        }
                        else{ //this starts at 100 and decreases brightness (v)
until there are no more secondary emotions
                              secHsv[1]+=svChange;
                        }
                  }
                  else{
                        secHsv[2]-=svChange; //descend from 100 brightness(v) to
about svLow
                  }
            }
            setPrimeColor(Color.getHSBColor(hsv[0]/360, hsv[1]/100, hsv[2]/100));
      }

      public SecondaryEmotion getSecondaryEmotion(String emote){
            for (SecondaryEmotion tempSec: secondary){
                  if (tempSec.getSecondary().equals(emote)){
                        return tempSec;
                  }
            }
            return null;
      }

      public String getPrimary() {
            return primaryString;
      }

      public SecondaryEmotion[] getSecondary() {
            return secondary;
      }

      public float[] getHSV() {
            return hsv;
      }

      public int getyTop() {
            return yTop;
      }

      public int getyBottom() {
            return yBottom;
      }

      public int getyCenter() {
            return yCenter;
      }
```

```
	public void setyCenter(int yCenter) {
		this.yCenter = yCenter;
	}

	public Color getPrimeColor() {
		return primeColor;
	}

	public void setPrimeColor(Color primeColor) {
		this.primeColor = primeColor;
	}
}
```

SecondaryEmotion.java

```
import java.awt.Color;


public class SecondaryEmotion {
	private String secondary;  //emotions that comprise/help define the primary
	private double weight; //weights to offset a word's unfair advantage
	private int[] hsv; //color for this Emotion
	private int yTop, yBottom, yCenter; //represent graphically the top and 
bottom of our image
	private Color secColor;

	public SecondaryEmotion(String secondary, double weight, int[] hsv, int yTop, 
int yBottom){
		this.secondary = secondary;
		this.weight = weight;
		this.hsv = hsv;
		this.yTop = yTop;
		this.yBottom = yBottom;
		setyCenter((Math.abs(yBottom-yTop)/2)+this.yTop);
		System.out.println("H:"+hsv[0]+" S:"+hsv[1]+" V:"+hsv[2]+" || 
Top/Bottom: "+getyTop()+"/"+getyBottom()+" ||Sec Y Center: "+getyCenter());
		setSecColor(Color.getHSBColor((float)(hsv[0]/360.0), (float)
(hsv[1]/100.0), (float)(hsv[2]/100.0)));
	}

	public String getSecondary() {
		return secondary;
	}

	public double getWeight() {
		return weight;
	}

	public int[] getHsv() {
		return hsv;
	}

	public int getyTop() {
		return yTop;
	}

	public int getyBottom() {
		return yBottom;
	}
	public String toString(){
		return secondary;
	}

	public int getyCenter() {
		return yCenter;
	}

	public void setyCenter(int yCenter) {
		this.yCenter = yCenter;
	}

	public Color getSecColor() {
		return secColor;
	}

	public void setSecColor(Color secColor) {
		this.secColor = secColor;
	}
}
```

Segment.java

```
import java.util.concurrent.ConcurrentHashMap;


public class Segment {
	private ConcurrentHashMap<String,Double> accumVals; //holds all values
	private MomentEmoteSpace[] newBounderies;
	private MomentEmotion[] momentAvgs; //averages for all thangs
	private Moment[] momentList; //list of all moments contained in this segment
	private int size, curSize, xTot, x; //size - how many moments this segment is
composed of / xStart-xEnd index
	private boolean full, empty; //when the segment has enough moments, it is
full (true)  /   empty is true when there are no moments in this segment
	private MomentEmoteSpace anchor;

	public Segment(int size){
		this.size = size;
		full = false;
		setEmpty(true);
		momentList = new Moment[size];
		accumVals = new ConcurrentHashMap<String,Double>();
		curSize = 0;
		xTot = 0;
	}

	public void addMoment(Moment newMoment){
		if (empty){
			momentAvgs = newMoment.getMomentSpectrum();
			empty = false;
		}
		if (!full){	//if the segment isn't full (has reached max number of
allowed Moment objects)
			momentList[curSize]=newMoment;	//the actual Moment object is
added to an array
			curSize++;	//curSize is increased
			MomentEmotion[] tempME = newMoment.getMomentSpectrum();
```

```
				for (int i=0; i<tempME.length;i++){ // for each primary
					momentAvgs[i].setPrimaryVal(momentAvgs[i].getPrimaryVal() + 
tempME[i].getPrimaryVal()); //add the primary value
					MomentEmoteSec[] secME = momentAvgs[i].getSecEmote(); //get 
secondaries
					System.out.println(secME.length);
					for (int j=0;j<secME.length;j++){ //add secondary value
						System.out.println(secME[j].getSecondaryString()+" 
"+tempME[i].getPrimaryString());
						secME[j].setSecondaryVal(secME[j].getSecondaryVal() + 
momentAvgs[i].getSecEmote()[j].getSecondaryVal()); // add the secondary val
					}
				}
				xTot+=newMoment.getX(); //x is added to to later be averaged
				if (curSize==size){ //if we have used all our space, we are now 
full
					full = true;
					createNewBounderies();
				}
			}
			else{
				System.out.println("Segement is full -- no moment added");
			}
		}

		public void sortMomentAvgs(){
			if (momentAvgs.length>1){
				for (int i=1;i<momentAvgs.length;i++){
					for (int j=i;j>0;j--){
						MomentEmotion tempEmote = momentAvgs[j];
						if (tempEmote.getPrimaryVal()>momentAvgs[j-
1].getPrimaryVal()){
							momentAvgs[j] = momentAvgs[j-1];
							momentAvgs[j-1] = tempEmote;
						}
					}
				}
			}
		}

		public void createNewBounderies(){
			newBounderies = new 
MomentEmoteSpace[EmotionSpectrum.getPrimaryStrings().length]; //list of new old 
sized boundaries created
			for (int i=0;i<EmotionSpectrum.getPrimaryStrings().length;i++){
				newBounderies[i] = new 
MomentEmoteSpace(getMomEmotion(EmotionSpectrum.getPrimaryStrings()[i]));  //list of 
old sized boundaries (to be changed) is populated;
			}
			//average all values with # of moments
			for (MomentEmotion curME: momentAvgs){
				double curMEVal = curME.getPrimaryVal()/curSize;
				curME.setPrimaryVal(curMEVal);
				for (MomentEmoteSec curSec: curME.getSecEmote()){
					curSec.setSecondaryVal(curSec.getSecondaryVal()/curSize);
				}
			}
			sortMomentAvgs(); //sort the momentemotespaces! based on most relevant
			x = xTot/curSize; //gets the average x
```

```
            int winPoints = 5;  //how many winners/points they get = how many
players there are
            int totalPoints = 0; //this will hold total points given out
            int curPoints = winPoints;
            for (MomentEmotion curME: momentAvgs){ //add all the points to their
respective spaces
                  getMomSpace(curME.getPrimaryString()).setPoints(curPoints);
                  totalPoints+=curPoints;
                  curPoints--;
                  if (curPoints==0){
                        break;
                  }
            }
            int onePointHeight = Visualize.getImageHeight()/totalPoints;
            int curBottomNewYTop = 0;
            int maxPoints = 0; //holds the max points
            anchor = newBounderies[0];
            //here we assign the Y changes
            for (int i=0; i<newBounderies.length;i++){ //for each boundary
                  int tempPoints = newBounderies[i].getPoints(); //we get the
amount of won points
                  System.out.println(newBounderies[i].getEmotion()+" TEMPPOINTS:
"+tempPoints);
                  if (tempPoints>maxPoints){
                        maxPoints = tempPoints;
                        anchor = newBounderies[i];
                  }
                  int threshold = 10;
                  double pointHeightChange =
getMomEmotion(newBounderies[i].getEmotion()).getPrimaryVal()/.05; //value that
makes onePointHeight worth less;
                  System.out.println("HEEEEY! "+newBounderies[i].getEmotion() +"
actual value "+ accumVals.get(newBounderies[i].getEmotion())+" ych
"+pointHeightChange + " "+onePointHeight);
                  if (pointHeightChange<1){ //if it is less than one, it is equal
to one
                        pointHeightChange=1;
                  }
                  else if (pointHeightChange>threshold){
                        pointHeightChange=threshold*.8;
                  }
                  System.out.println("point height change "+pointHeightChange+" "+
(onePointHeight*pointHeightChange)*tempPoints);
                  curBottomNewYTop = (int)((newBounderies[i].getTop()+
(((onePointHeight/(threshold-pointHeightChange))*tempPoints)))); //we get a new
yBottom based on points
                  System.out.println("tot "+curBottomNewYTop);
                  newBounderies[i].setBottom(curBottomNewYTop);  //we set a new
bottom
                  if (i!=newBounderies.length-1){ //if we are not on the last entry
                        newBounderies[i+1].setTop(curBottomNewYTop);
                  }
            }
            shiftBounderies();
            for (MomentEmoteSpace momEmo: newBounderies){
                  momEmo.assignSecondaries();  //this assigns our secondary y-
bounds based on their relative value to each other
            }
      }
```

```
	public MomentEmotion getMomEmotion(String primary){
		for (MomentEmotion curMomEmo: momentAvgs){
			if (curMomEmo.getPrimaryString().equals(primary)){
				return curMomEmo;
			}
		}
		System.out.println("THIS IS MESSED UPPPPPPPP\n\n\n\n\n");
		return null;
	}

	public MomentEmoteSpace getMomSpace(String primary){
		for (MomentEmoteSpace curMom: newBounderies){
			if (curMom.getEmotion().equals(primary)){
				return curMom;
			}
		}
		return null;
	}

	/**
	 * Shifts the y points based on the anchor
	 */
	public void shiftBounderies(){
		int origTop =
EmotionSpectrum.getPrimary(anchor.getEmotion()).getyTop();
		int curTop = anchor.getTop();
		int curBottom = anchor.getBottom();
		int origBottom =
EmotionSpectrum.getPrimary(anchor.getEmotion()).getyBottom();
		int origDist = origBottom-origTop;
		int curDist = curBottom-curTop;
		int fringeDist = (curDist - origDist)/2; //get the fringe distance
		int newTop = origTop - fringeDist;
		int travelAll=newTop - origTop;
		System.out.println(travelAll);
		for (MomentEmoteSpace tempSpace: newBounderies){
			System.out.println("SETTOP: "+(tempSpace.getTop()+travelAll));
			tempSpace.setTop(tempSpace.getTop()+travelAll);
			tempSpace.setBottom(tempSpace.getBottom()+travelAll);
		}
	}

	/**
	 * Adds to count of wins this emotion has
	 */
	public void addMomentEmoteWinCount(String emote, int points){
		for (MomentEmoteSpace tempSpace: newBounderies){
			if (tempSpace.getEmotion().equals(emote)){
				tempSpace.setPoints(tempSpace.getPoints()+points);
				return;
			}
		}
	}

	public MomentEmoteSpace[] getBounderies(){
		if (!full){
			createNewBounderies();
		}
```

```
            return newBounderies;
        }

        public boolean isFull(){
            return full;
        }

        public boolean isEmpty() {
            return empty;
        }

        public void setEmpty(boolean empty) {
            this.empty = empty;
        }

        public int getX() {
            return x;
        }

        public void setX(int x) {
            this.x = x;
        }
}
```

TopicModel.java

```
import cc.mallet.types.*;
import cc.mallet.pipe.*;
import cc.mallet.pipe.iterator.*;
import cc.mallet.topics.*;

import java.util.*;
import java.util.regex.*;
import java.io.*;

public class TopicModel  {

    public static void topicModel(String name) throws Exception {
        //Begin by importing documents from text to feature sequences
        ArrayList<Pipe> pipeList = new ArrayList<Pipe>();
        // Pipes: lowercase, tokenize, remove stopwords, map to features
        pipeList.add( new CharSequenceLowercase() );
        pipeList.add( new CharSequence2TokenSequence(Pattern.compile("\\p{L}[\\p{L}\\p{P}]
+\\p{L}")) );
        pipeList.add( new TokenSequenceRemoveStopwords(new File("referenceFiles/en.txt"), "UTF-8",
false, false, false) );
        pipeList.add( new TokenSequence2FeatureSequence() );
        InstanceList instances = new InstanceList (new SerialPipes(pipeList));
        System.getProperty("user.dir");
        File file = new File("books/"+name+".txt");
        Reader fileReader = new InputStreamReader(new FileInputStream(file), "UTF-8");
        instances.addThruPipe(new CsvIterator (fileReader, Pattern.compile("^(\\S*)[\\s,]*(\\S*)[\\s,]*(.*)
$"),
                                        3, 2, 1)); // data, label, name fields

        // Create a model with 100 topics, alpha_t = 0.01, beta_w = 0.01
        //  Note that the first parameter is passed as the sum over topics, while
        //  the second is the parameter for a single dimension of the Dirichlet prior.
        int numTopics = 5;
        ParallelTopicModel model = new ParallelTopicModel(numTopics, 1.0, 0.01);

        model.addInstances(instances);

        // Use two parallel samplers, which each look at one half the corpus and combine
        //  statistics after every iteration.
        model.setNumThreads(2);

        // Run the model for 50 iterations and stop (this is for testing only,
        //  for real applications, use 1000 to 2000 iterations)
        model.setNumIterations(1000);
        model.estimate();

        // Show the words and topics in the first instance

        // The data alphabet maps word IDs to strings
        Alphabet dataAlphabet = instances.getDataAlphabet();

        FeatureSequence tokens = (FeatureSequence) model.getData().get(0).instance.getData();
        LabelSequence topics = model.getData().get(0).topicSequence;

        Formatter out = new Formatter(new StringBuilder(), Locale.US);
        for (int position = 0; position < tokens.getLength(); position++) {
            out.format("%s-%d ", dataAlphabet.lookupObject(tokens.getIndexAtPosition(position)),
topics.getIndexAtPosition(position));
        }
        System.out.println(out);

        // Estimate the topic distribution of the first instance,
        //  given the current Gibbs state.
        double[] topicDistribution = model.getTopicProbabilities(0);

        // Get an array of sorted sets of word ID/count pairs
        ArrayList<TreeSet<IDSorter>> topicSortedWords = model.getSortedWords();

        // Show top 5 words in topics with proportions for the first document
        for (int topic = 0; topic < numTopics; topic++) {
            Iterator<IDSorter> iterator = topicSortedWords.get(topic).iterator();

            out = new Formatter(new StringBuilder(), Locale.US);
            out.format("%d\t%.3f\t", topic, topicDistribution[topic]);
            int rank = 0;
            while (iterator.hasNext() && rank < 20) {
                IDSorter idCountPair = iterator.next();
                out.format("%s (%.0f) ", dataAlphabet.lookupObject(idCountPair.getID()),
idCountPair.getWeight());
                rank++;
            }
            System.out.println("t1 "+out);
        }

        // Create a new instance with high probability of topic 0
        StringBuilder topicZeroText = new StringBuilder();
        Iterator<IDSorter> iterator = topicSortedWords.get(0).iterator();

        int rank = 0;
        while (iterator.hasNext() && rank < 5) {
            IDSorter idCountPair = iterator.next();
            topicZeroText.append(dataAlphabet.lookupObject(idCountPair.getID()) + " ");
            rank++;
        }

        // Create a new instance named "test instance" with empty target and source fields.
        InstanceList testing = new InstanceList(instances.getPipe());
        testing.addThruPipe(new Instance(topicZeroText.toString(), null, "test instance", null));

        TopicInferencer inferencer = model.getInferencer();
        double[] testProbabilities = inferencer.getSampledDistribution(testing.get(0), 10, 1, 5);
        System.out.println("0\t" + testProbabilities[0]);
    }

    public static void main (String args[]) throws Exception{
            TopicModel.topicModel("ulys");
    }
}
```

Visualize.java

```
import processing.core.*;
import java.util.ArrayList;
import java.util.LinkedList;
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.GradientPaint;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.LinearGradientPaint;
import java.awt.Paint;
import java.awt.RenderingHints;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.GeneralPath;
import java.awt.geom.Path2D;
import java.awt.geom.Point2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.Timer;

public class Visualize extends JPanel {
	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	private static ArrayList<int[]> plotPoints;
	private static ArrayList<Moment> allMoments;
	private static ArrayList<Segment> novelSegments;
	private ArrayList<ArrayList<double[]>> shapePoints;
	private ArrayList<ArrayList<double[]>> shapeLines;
	private static ArrayList<String[]> plotTexts;
	private static Segment curSegment;
	private static Novel novelObj;
	private static int height, width, textLength, segSize;
	private static boolean primaryOnly;

	public Visualize(){
		primaryOnly = false;
		height = 700;
		width = 1000;
		int marginTop = 100;
		segSize = 2;
		setSize(height,width);
		curSegment = new Segment(segSize);
		novelSegments = new ArrayList<Segment>();
		allMoments = new ArrayList<Moment>();
		plotPoints = new ArrayList<int[]>();
		plotTexts = new ArrayList<String[]>();
		shapePoints = new ArrayList<ArrayList<double[]>>();
		novelObj = new Novel("huckfin", "testAuthor", (int)(height),marginTop); //Novel class
initialized
		textLength = novelObj.getTextLength();
		populatePointLists();
		saveImage();
	}
	public void populatePointLists(){
		System.out.println("POPULATING");
		if (primaryOnly){ //only shows primary colors
			for (int i=0;i<novelSegments.size();i++){  //for each segment
				 MomentEmoteSpace[]
bounderies=novelSegments.get(i).getBounderies();
				 for (int j=0;j<bounderies.length;j++){ //for each category
					if (i==0){  //if this is our first time through, we create all the point
array lists
						shapePoints.add(new ArrayList<double[]>()); //add an
array list to be added to!
					}
					System.out.println(bounderies[j].getEmotion());
					double[] pointXYTemp = {((novelSegments.get(i).getX()*width)/
(textLength)),bounderies[j].getTop()}; //create temporary xy point
					shapePoints.get(j).add(pointXYTemp); //add this point to this
category's list
				}
			}
			for (int i=novelSegments.size()-1;i>=0;i--){  //for each segment
				MomentEmoteSpace[]
bounderies=novelSegments.get(i).getBounderies();
				for (int j=0;j<bounderies.length;j++){ //for each category
					double[] pointXYTemp = {((novelSegments.get(i).getX()*width)/
(textLength)),bounderies[j].getBottom()}; //create temporary xy point
					shapePoints.get(j).add(pointXYTemp); //add this point to this
category's list
				}
			}
		}
		else{ //shows secondary colors
			for (int i=0;i<novelSegments.size();i++){  //for each segment
				int curSec = 0;
				 MomentEmoteSpace[]
bounderies=novelSegments.get(i).getBounderies();
				 for (int j=0;j<bounderies.length;j++){ //for each primary
					 MomentEmoteSpace[] secondaries =
bounderies[j].getSecondary();
					 for (int k=0;k<secondaries.length;k++){ //for each secondary
						if (i==0){  //if this is our first time through, we create all
the point array lists
							shapePoints.add(new ArrayList<double[]>()); //add
an array list to be added to!
						}
						System.out.println(secondaries[k].getEmotion());
						double[] pointXYTemp =
{((novelSegments.get(i).getX()*width)/(textLength)),secondaries[k].getTop()}; //create temporary xy
point
						shapePoints.get(curSec).add(pointXYTemp); //add this
point to this category's list
                                                    curSec++;
                                            }
                                            System.out.println(i+" JSIZE: "+shapePoints.get(j).size());
                                    }
                            }
                            for (int i=novelSegments.size()-1;i>=0;i--){  //for each segment
                                    int curSec = 0;
                                    MomentEmoteSpace[]
bounderies=novelSegments.get(i).getBounderies();
                                     for (int j=0;j<bounderies.length;j++){ //for each primary
                                            MomentEmoteSpace[] secondaries =
bounderies[j].getSecondary();
                                            for (int k=0;k<secondaries.length;k++){ //for each secondary
                                                    double[] pointXYTemp =
{((novelSegments.get(i).getX()*width)/(textLength)),secondaries[k].getBottom()}; //create temporary
xy point
                                                    shapePoints.get(curSec).add(pointXYTemp); //add this
point to this category's list
                                                    curSec++;
                                            }
                                    }
                            }
                    }
            }

            public void paint(Graphics g) {
                    super.paint(g);
                    System.out.println();
        Graphics2D g2d = (Graphics2D)g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                    RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setRenderingHint(RenderingHints.KEY_RENDERING,
                    RenderingHints.VALUE_RENDER_QUALITY);
        //create shapes from point lists
        System.out.println("ShapePointsSize: "+shapePoints.size());
        GeneralPath[] shapes = new GeneralPath[shapePoints.size()];

        g2d.setBackground(Color.white);
        if (primaryOnly){
                            for (int i=0;i<shapePoints.size();i++){
                                    shapes[i] = new GeneralPath();
                                    ArrayList<double[]> shapePointsCat = shapePoints.get(i);
                                    shapes[i].moveTo(shapePointsCat.get(0)[0], shapePointsCat.get(0)
[1]); //move to the first point
                                    for (int j=0;j<shapePoints.get(i).size();j++){
                                            shapes[i].lineTo(shapePointsCat.get(j)[0],shapePointsCat.get(j)
[1]);
                                    }
                                    shapes[i].closePath();
                                    g2d.setPaint(new GradientPaint(0,0,EmotionSpectrum.getSpectrum()
[i].getPrimeColor(),1000,1000,EmotionSpectrum.getSpectrum()[(i==(shapePoints.size()-1)?
i:i+1)].getPrimeColor()));
                                    g2d.fill(shapes[i]);
                                    g2d.draw(shapes[i]);
                              }
        }
        else{
           ArrayList<Color> secColors = new ArrayList<Color>();
           for (PrimaryEmotion prime: EmotionSpectrum.getSpectrum()){
                  for (SecondaryEmotion sec: prime.getSecondary()){
                        secColors.add(sec.getSecColor());
                  }
           }
           for (int i=0;i<shapePoints.size();i++){ //for each shape
                  System.out.println(i);
                                    shapes[i] = new GeneralPath();
                                    ArrayList<double[]> shapePointsCat = shapePoints.get(i);
                                    shapes[i].moveTo(shapePointsCat.get(0)[0], shapePointsCat.get(0)
[1]); //move to the first point
                                    for (int j=0;j<shapePointsCat.size();j++){ //for each
                                          shapes[i].lineTo(shapePointsCat.get(j)[0],shapePointsCat.get(j)
[1]);
                                    }
                                    shapes[i].closePath();
                                    g2d.setPaint(new
GradientPaint(0,0,secColors.get(i),1000,1000,secColors.get(i==(shapePoints.size()-1)?i:i+1)));
                                    g2d.fill(shapes[i]);
                                    g2d.draw(shapes[i]);
                              }
        }
    }


           public static void getPlotsToAdd(){
                  LinkedList<String> castList = novelObj.getListOfCharacters();
                  for (String tempActor: castList){
                        ArrayList<Moment> tempMomentList =
novelObj.getCharacter(tempActor).getEmotionalRollercoaster();
                        for (Moment tempMoment: tempMomentList){
                              System.out.println("\n\n--------"+tempMoment.getWord()+"---------");
                              for (MomentEmotion tempMoEmote:
tempMoment.getMomentSpectrum()){
                                    for (MomentEmoteSec tempMoSec:
tempMoEmote.getSecEmote()){
                                          System.out.println(tempMoSec.getSecondaryString()+"---
X: "+(tempMoment.getX()*width)/textLength+"  Y: "+tempMoSec.getY()+ "  Size:
"+tempMoSec.getSize());
```

```
							//addPlotPoint((int)
(((tempMoment.getX()*width)/textLength)), (int)(tempMoSec.getY()), tempMoSec.getSize(),
EmotionSpectrum.getPrimary(tempMoEmote.getPrimaryString()).getSecondaryEmotion(tempMoSec.g
etSecondaryString()).getHsv());
						}
					}
				}
			}
		}

		public static void drawSegments(){
			if (!curSegment.isEmpty()){
				novelSegments.add(curSegment);
			}
			int[] hsvTemp = {360,100,100};  //temporarily used to whatever I will remove it soon
			System.out.println("SIZEEEE:"+novelSegments.size());
			for (Segment tempSeg: novelSegments){
				if (!tempSeg.isEmpty()){
					for (MomentEmoteSpace tempSpace: tempSeg.getBounderies()){
						//addPlotPoint(((tempSeg.getX()*width)/(textLength*2)),
tempSpace.getTop(), 2, hsvTemp);
						//addPlotPoint(((tempSeg.getX()*width)/(textLength*2)),
tempSpace.getBottom(), 2, hsvTemp);
					}
				}
			}
		}

		public static void addToAllMoments(Moment newMoment){
			allMoments.add(newMoment);
			if (curSegment.isFull()){
				novelSegments.add(curSegment);
				curSegment = new Segment(segSize);
			}
			else{
				curSegment.addMoment(newMoment);
			}
		}

		/**
		 * Stumbled upon a previously defined getHeight(); shoulda known...processing!
		 * @return
		 */
		public static int getImageHeight(){
			return height;
		}

		public void saveImage(){
			BufferedImage bi = new BufferedImage(this.getSize().width+1000,
this.getSize().height+1000, BufferedImage.TYPE_INT_ARGB);
            Graphics g = bi.createGraphics();
            this.paint(g);  //this == JComponent
            g.dispose();
            try{ImageIO.write(bi,"png",new File("test.png"));}catch (Exception e) {}
    }

    public static void main(String args[]) {
            JFrame frame = new JFrame("Visualization");
        frame.add(new Visualize());
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(1000, 700);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }
}
```

Word.java

```
import java.util.LinkedList;

public class Word {
	private String word;
	private LinkedList occurences;
	private int recentIndex;

	public Word(String word, int wordInd, String pOS){
		System.out.println(word);
		occurences = new LinkedList();
		addOccurence(wordInd);
		this.word = word;
	}

	public void addOccurence(int i){
		recentIndex = i;
		occurences.add(i);
	}

	public LinkedList getIndex(){
		return occurences;
	}

	public int getRecentIndex(){
		return recentIndex;
	}

	public String getWord(){
		return word;
	}

	public int getCount(){
		return occurences.size();
	}
}
```