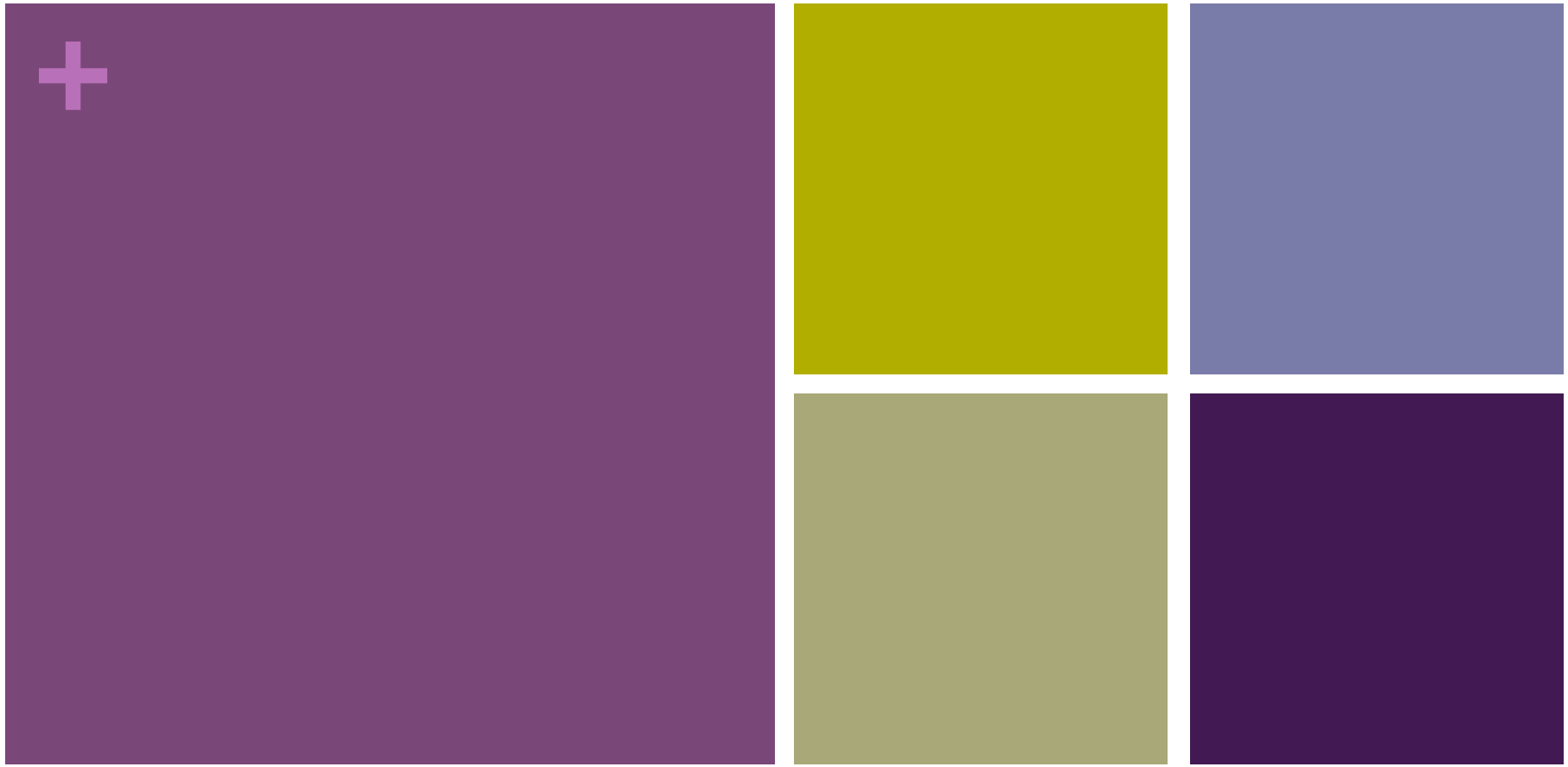


Word Clouds Implementation

Text Processing

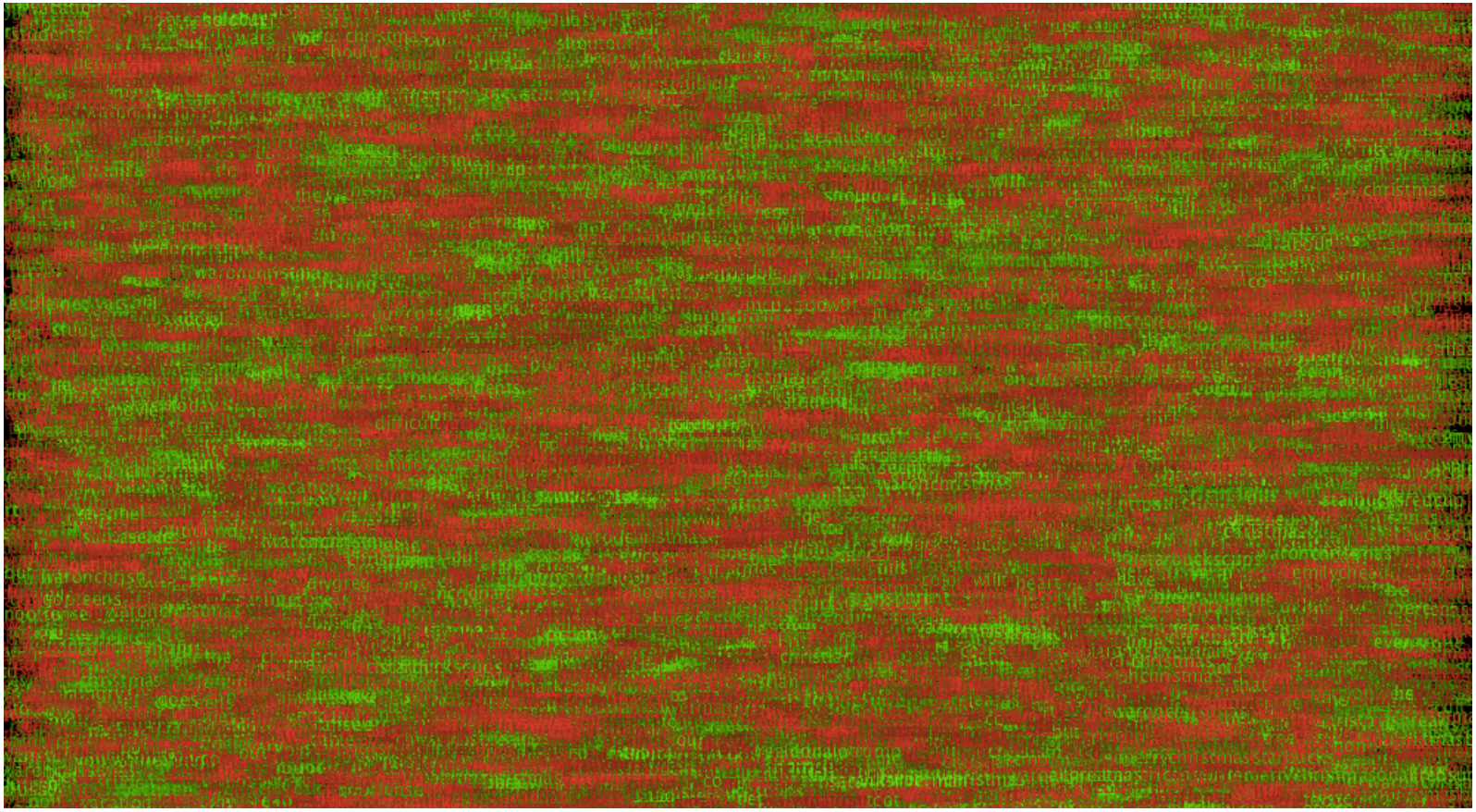
Data Visualization Process

- Acquire - Obtain the data from some source
- Parse - Give the data some structure, clean up
- Filter - Remove all but the data of interest
- Mine - Use the data to derive interesting properties
- Represent - Chose a visual representation
- Refine – Improve to make it more visually engaging
- Interact - Make it interactive

Text Visualization

- Source = Document
- Parse = Words
- Filter = Word Set with counts
- Mine = Get relevant words
- Represent = Fonts/Placement
- Refine/Interact

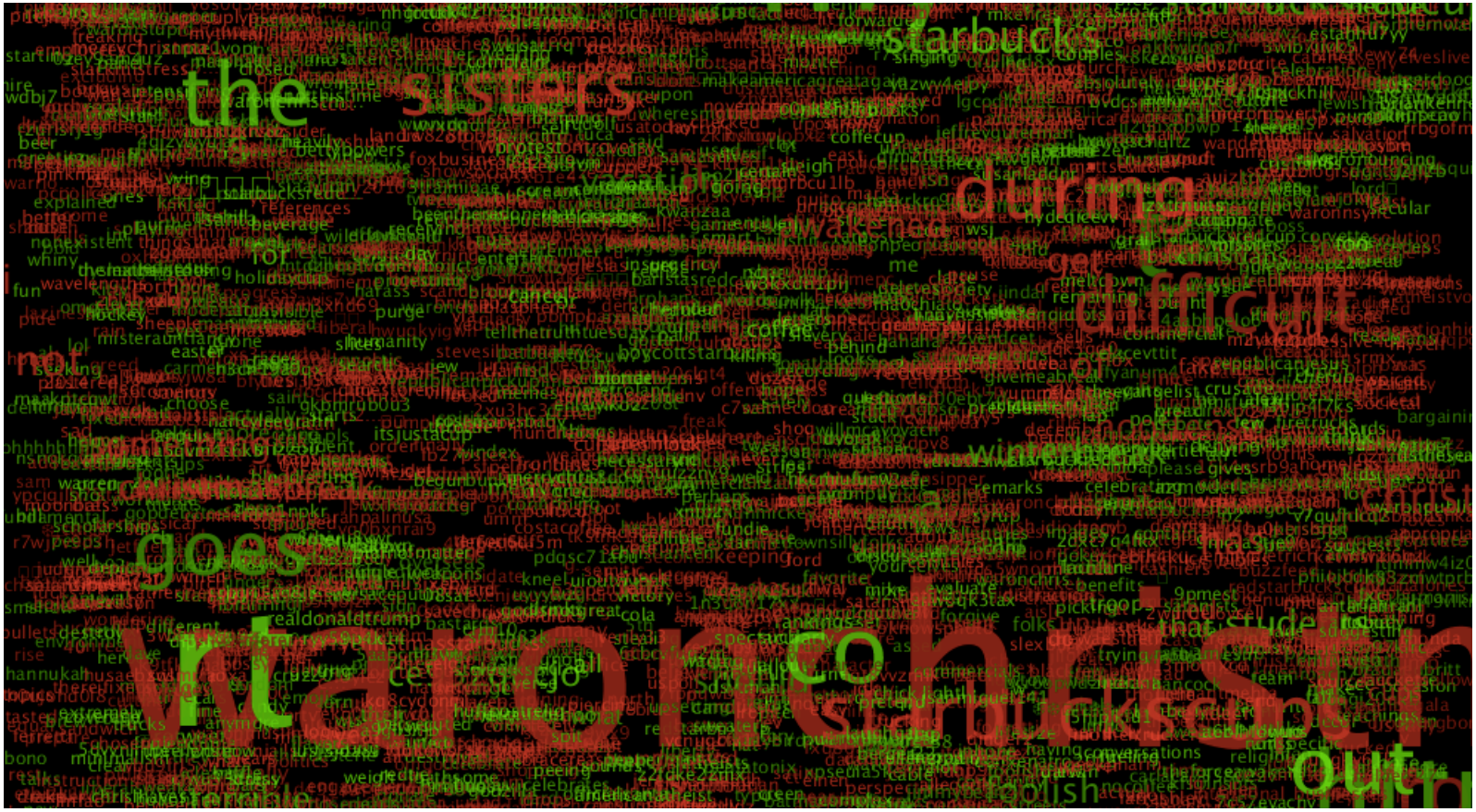
Displaying: Step 1 show words



Filtering: Word Frequency List

- Create a set of word frequency pairs.
- Algorithm:
  - create empty set pairs
  - for each token
    - if pairs has (token,count)
      - increment count
    - otherwise
      - add (token, 1)
- We did this with an ArrayList
- We also did this with a HashMap

Displaying: step 2 size words



Displaying: step 3 reduce number using Sorted Array of words

Displaying: step 4 reduce number of words

Other Filtering

- Stopwords
  - compare tokens with an array of stopwords, make a subset of tokens that has no stopwords.
- hastag removal
  - if(token[i].charAt(0) == '#') { // if it's a hashtag…
- topic words
  - only display words that are about a particular topic using a list or multiple lists of keepwords
- substring filter
  - remove or keep a word that contains a substring
  - if(token[i].contains("fun") { // if fun is in the word

Stopwords Algorithm

- read array of stopwords
- create array of filteredWords
- count = 0
- for each token t
  - boolean add = true
  - for each stopword s
    - if s.equals(t)
      - add = false
  - if add
    - filteredWords[count] = t;
    - increment count

Hashtag Removal Algorithm

- create array of filteredWords
- count = 0
- for each token t
  - if(token[i].charAt(0) != '#')
    - filteredWords[count] = t;
    - increment count

Topic words keep Algorithm

- read array of topic words
- create array of filteredWords
- count = 0
- for each token t
  - boolean add = false
  - for each topic word s
    - if s.equals(t)
      - add = true
  - if add
    - filteredWords[count] = t;
    - increment count

Substring filter keep Algorithm

- read array of substrings
- create array of filteredWords
- count = 0
- for each token t
  - boolean add = false
  - for each substring s
    - if t.contains(s)
      - add = true
  - if add
    - filteredWords[count] = t;
    - increment count

Arrange

- Non-overlapping arrangements are often desired
  - a.k.a. Tiling
- Make a Word Tile Object
  - holds the word, frequency pair
  - displays itself
  - should have a concept of visual intersection
- How do we arrange?
  - randomly?
  - grid?
  - spiral?

Random Arrangement

- While there are more tiles to place
  - get the next tile, t, to place
  - while(t is not placed)
    - set a random location, l, for the tile
    - if t does not intersect any previously placed tile
      - place t.

checking t against previously placed tiles

- basic idea
  - keep the index of the current item to place
  - randomly place the item at current index
  - loop from 0 to the current index and check if the place intersects
  - if not then increment current index
- details
  - for (int j = 0; j < sortedList.size(); j++)
    - while goodPlace == false
      - randomly place sortedList.get(j)
      - goodPlace = true
      - for(int i = 0; i < j; i++) {
        - if sortedList.get(i).intersects(sortedList.get(j))
          - goodPlace = false

Grid arrangement (simplest way)

- Get the size of the biggest tile.
- compute how many of the biggest tile would fit in the window
- make a grid of width/tileWidth x height/tileHeight words each scaled based on their frequency.

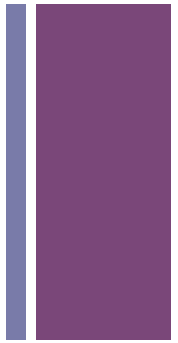
Grid arrangement (slightly tougher way)

- Get the size of the biggest tile.
- compute how many, M, of the biggest tile would fit in the sketch
- if N > M, then change the maximum font size of a tile so that a grid of the largest tile size would allow for N tiles on the sketch
- make a grid based on new tile sizes.

Spiral Arrangement

- Sort the tiles from largest to smallest.
- While there are more tiles to place
  - get the next tile, t, to place
  - while(t is not placed)
    - set location, l, for the tile to be at the current spiral location
    - if t does not intersect any previously placed tile
      - place t.
    - update the current spiral position outward by a fixed step size.

Let's look at some code



- warOnChristmas_v1b
- warOnChristmas_v1c

Task

- get in groups of 3 or 4
- create a secondary filter so that your words have more meaning
- create a tiling of your choosing so that there is no overlap.