```
import twitteroauth
from muchtwitter.models.twitter import TweetProcessor
from gui import Gui
from muchtwitter.models.logs import Logger
from muchtwitter.models.document import Document, DocQueue, WordDictionary
import time

QUERY_FREQ = 5000

# SEARCH_TERM = "Chicago"
logger = Logger()
api = twitteroauth.getAuthenticatedApi()
tweetprocessor = TweetProcessor()
gui = None
docWords = DocQueue(10)
wordDict = WordDictionary()


def searchEvent():

    SEARCH_TERM = get_string()
    print SEARCH_TERM
    if len(SEARCH_TERM) > 0:
        results = api.GetSearch(SEARCH_TERM, lang="en")

        start_time = time.time()
        resultString = ""
        for result in results:
            logger.logTweet(result)
            resultString+= ' ' + result.text
        resultString.lower()
        print resultString.encode('utf-8')
        document = Document(resultString)
        # Go through each word list
        for word in wordDict.getHappy():
            tfIdf = docWords.calcTfIdf(word, 'happy', document)
        for word in wordDict.getSad():
            tfIdf = docWords.calcTfIdf(word, 'sad', document)
        for word in wordDict.getAngry():
            tfIdf = docWords.calcTfIdf(word, 'angry', document)
        for word in wordDict.getProfane():
            tfIdf = docWords.calcTfIdf(word, 'profane', document)
        # Add the document to our deque
        docWords.addDoc(document)
        tweetprocessor.processWeights(docWords)#wordList)
        logger.logTiming("tfIdf", (time.time() - start_time), tweetprocessor.calcHighest())

        countAndColor()

def countAndColor():
    highest = tweetprocessor.calcHighest()
    print '\n\n Highest emotion: ' + highest + '\n\n'
    print 'Accumulated so far: '
    print tweetprocessor.counts
    # logger.logCounts(tweetprocessor.counts)

    print '\n\n'
    if (highest == 'happy'):
        gui.setColor('green')
    elif (highest == 'angry'):
        gui.setColor('red')
    elif (highest == 'sad'):
        gui.setColor('blue')
    elif (highest == 'profane'):
        gui.setColor('orange')
    # Do process again after 15 sec
    _job = gui.after(QUERY_FREQ, searchEvent)

def quitCallback():
    print "Exited."
    gui.stopGui()

def get_string():
    search_text = gui.search.get()
    if len(search_text) > 0 or search_text is not Nil:
        return search_text
    else:
        return Nil

if __name__ == '__main__':
    gui = Gui(quitCallback,get_string)
    _job = gui.after(QUERY_FREQ, searchEvent)
    gui.mainloop()
```

```python
import twitter
import getpass
from settings import *



"""
Gets an authenticated API for MuchMoodyTweet app
"""
def getAuthenticatedApi():
    # get settings from the settings.py file
    apiKey = API_KEY
    accessToken = ACCESS_TOKEN
    accessSecret = ACCESS_SECRET
    apiSecret = API_SECRET

    api = twitter.Api(consumer_key=apiKey,
                      consumer_secret=apiSecret,
                      access_token_key=accessToken,
                      access_token_secret=accessSecret)
    return api
```

```
#!/usr/bin/env python
# encoding: utf-8

"""
gui.py

A simple GUI that can display colors
"""

import Tkinter as tk
import sys
# from main import searchEvent

searchText = "hello"

class Gui(tk.Frame):
    def __init__(self, quitCallback, get_string, master=None):
        tk.Frame.__init__(self, master)
        self.master.title('Much Twitter')
        self.quitCallback = quitCallback
        self.get_string = get_string
        self.color = 'white'
        #TODO do we need grid? self.grid(column=0, row=0, ipadx=100, ipady=100, sticky=('N','W
','E','S'))
        self.configure(background=self.color)
        self.grid_rowconfigure(0, weight=1)
        self.grid_columnconfigure(0, weight=1)
        self.pack(fill="both", expand=True, ipadx=100, ipady=100)
        self.createWidgets()


    def stopGui(self):
        self.destroy()
        sys.exit(0)

    def createWidgets(self):
        #TODO text box for search query text

        self.search = tk.Entry(self)
        self.search.grid()

        self.submit_button = tk.Button(self, text="Submit", command=self.get_string)
        self.submit_button.grid()

        b = self.quitButton = tk.Button(self, text='Quit', command=self.quitCallback)
        self.quitButton.grid()

    def setColor(self, color):
        self.color = color
        self.configure(background=self.color)



""" Example execution """
#app = Gui()
#app.mainloop()
```

```
class TweetProcessor(object):
    "Class for dealing with tweets retrieve from the API"

    def __init__(self):
        # Master sums
        self.counts = {'happy': 0.0, 'angry': 0.0, 'sad': 0.0, 'profane': 0.0, 'happyEm': 0.0,
 'angryEm': 0.0, 'sadEm': 0.0}

    #def readTweetLogs(self, log):
        #TODO

    def processTweet(self, tweet):
        self.counts['happy'] += tweet.happyWordCount
        self.counts['angry'] += tweet.angryWordCount
        self.counts['sad'] += tweet.sadWordCount
        self.counts['profane'] += tweet.profaneWordCount
        self.counts['happyEm'] += tweet.happyEmoticonCount
        self.counts['angryEm'] += tweet.angryEmoticonCount
        self.counts['sadEm'] += tweet.sadEmoticonCount

    def processWeights(self, docWords):#realWordList):
        self.clearCounts()
        for doc in docWords.getDocQueue():
            for w, rWord in doc.getWordDict().iteritems():
                #print 'WORD WEIGHT AFTER: {}'.format(rWord.getWeight())
                if(rWord.getEmo() == "happy"):
                    self.counts['happy'] += rWord.getWeight()
                if(rWord.getEmo() == "angry"):
                    self.counts['angry'] += rWord.getWeight()
                if(rWord.getEmo() == "sad"):
                    self.counts['sad'] += rWord.getWeight()
                if(rWord.getEmo() == "profane"):
                    self.counts['profane'] += rWord.getWeight()

    def calcHighest(self):
        highest = max(self.counts.iterkeys(), key=(lambda key: self.counts[key]))
        return highest

    def clearCounts(self):
        self.counts['happy'] = 0.0
        self.counts['angry'] = 0.0
        self.counts['sad'] = 0.0
        self.counts['profane'] = 0.0
        self.counts['happyEm'] = 0.0
        self.counts['angryEm'] = 0.0
        self.counts['sadEm'] = 0.0
```

```
import sqlite3


class Database():
    HAPPY_WORDS = 'happywords'
    SAD_WORDS = 'sadwords'
    PROFANE_WORDS = 'profane'
    ANGRY_WORDS = 'angrywords'

    query = 'SELECT * FROM %s'

    def __init__(self):
        self.conn = sqlite3.connect('data')
        self.c = self.conn.cursor()

    def get_happywords(self):
        words = [row[0] for row in self.c.execute(Database.query % Database.HAPPY_WORDS)]
        return words

    def get_sadwords(self):
        words = [row[0] for row in self.c.execute(Database.query % Database.SAD_WORDS)]
        return words

    def get_profanewords(self):
        words = [row[0] for row in self.c.execute(Database.query % Database.PROFANE_WORDS)]
        return words

    def get_angrywords(self):
        words = [row[0] for row in self.c.execute(Database.query % Database.ANGRY_WORDS)]
        return words
```

```
from collections import deque
import re
from settings import *

PRECISION = 5

class Document(object):
    """An object that stores a long string of Tweets to form a document. Also contains a dicit
onary of words that have been found within the document with a count"""
    def __init__(self, docString):
        self.doc = docString
        self.wordDict = dict()

    def getDocString(self):
        return self.doc

    def setDocString(self, docString):
        self.doc = docString

    def getWordDict(self):
        return self.wordDict

    " Adds a word to our word dict list, replacing the old if it exists "
    def addRealWord(self, word):
        self.wordDict[word.name] = word

    def hasWord(self, word):
        return word in self.wordDict



class DocQueue(object):
    """An object that stores a duque of Documents and list of Words and calculates the TF/IDF"
""
    def __init__(self, maxWindow):
        self.docs = deque('', maxWindow)

    def getDocQueue(self):
        return self.docs

    " Adds a document to the right side of the deque, pushing off the left if maxWindow is exc
eeded. "
    def addDoc(self, doc):
        self.docs.append(doc)

    # Calculates the TF of a word, adds the count of the word to the doc, and stores it in our
 deque.
    # It then uses the current deque state to calculate IDF, returning the weight.
    # Returns the document that should be added
    def calcTfIdf(self, word, emotion, document):
        TF = 0.0
        IDF = 0.0
        # Use the document and find each word, counting to calc TF
        docString = document.getDocString()
        allWords = re.split(r'\W', docString)
        allWordCount = len(allWords)
        thisWordCount = allWords.count(word)
        if(thisWordCount > 0):
            print 'FOUND {} {} words'.format(thisWordCount, emotion)
        # Calc the TF
        TF = round(thisWordCount / float(allWordCount), PRECISION)
        if(thisWordCount > 0):
            print 'TF CALC: {} / {} = {}'.format(thisWordCount, allWordCount, TF)
        # Calc the IDF
        docsWithWord = 0
        if(thisWordCount > 0):
            docsWithWord = 1
        for doc in self.docs:
            if(doc.hasWord(word)):
                docsWithWord += 1
        IDF = 0
        if(docsWithWord == 0):
            IDF = 0.0
        else:
            IDF = round((len(self.docs) + 1) / float(docsWithWord), PRECISION)
        if(thisWordCount > 0):
            print 'IDF CALC: {} / {} = {}'.format(len(self.docs), docsWithWord, IDF)
        # Calc the TfIdf
        TfIdf = round(TF * IDF, PRECISION)
        if(thisWordCount > 0):
            print 'WEIGHT: {}'.format(TfIdf)
        realWord = Word(word)
        realWord.setTF(TF)
        realWord.setIDF(IDF)
        realWord.setWeight(TfIdf)
        realWord.setEmo(emotion)
        document.addRealWord(realWord)
        return document



class Word(object):
    """An object that takes the place of a TF/IDF word, containing its TF, IDF, name, and calc
 weight"""
    def __init__(self, name):
        self.name = name
        self.TF = 0.0
        self.IDF = 0.0
        self.Weight = 0.0
        self.emo = ""

    def getName(self):
        return self.name

    def getTF(self):
        return self.TF

    def getIDF(self):
        return self.IDF

    def getWeight(self):
        return self.Weight

    def getEmo(self):
        return self.emo

    def setName(self, name):
        self.name = name

    def setTF(self, TF):
        self.TF = TF

    def setIDF(self, IDF):
        self.IDF = IDF

    def setEmo(self, emo):
        self.emo = emo

    def setWeight(self, weight):
        self.Weight = weight

class WordDictionary(object):
    "Make all the dictionaries available in array form"

    def __init__(self):
        self.happyWords = [line.strip() for line in open(DICTIONARY_DIR + '/' + HAPPY_WORDS)]
        self.sadWords = [line.strip() for line in open(DICTIONARY_DIR + '/' + SAD_WORDS)]
        self.angryWords = [line.strip() for line in open(DICTIONARY_DIR + '/' + ANGRY_WORDS)]
        self.profaneWords = [line.strip() for line in open(DICTIONARY_DIR + '/' + PROFANE_WORD
S)]

    def getHappy(self):
        return self.happyWords

    def getSad(self):
        return self.sadWords

    def getAngry(self):
        return self.angryWords

    def getProfane(self):
        return self.profaneWords

    def getAll(self):
        return self.happyWords + self.sadWords + self.angryWords + self.profaneWords
```

```
#!/usr/bin/env python
# encoding: utf-8


from threading import Thread # This is the right package name
import ctypes
from utils import Utility
import csv
import codecs
import string
import random
import time
import datetime
import threading
import sys, os



class Logger(object):
    "Class for logging tweet data"
    def __init__(self, messageFileName="default"):
        self.messageFileName = messageFileName
        self.messageLogger = csv.writer(open( self.messageFileName, 'wb'))
        self.messageLogger.writerow(["time", "code", ""])

        self.tweetLogger = csv.writer(open("tweets.csv", 'a'))

        self.util = Utility()
        self.time = self.util.currentTimeSeconds()

        self.timeLogger = csv.writer(open("timing.csv", 'wb'))
        self.timeLogger.writerow(['impl', 'execTime', 'highestEmo', 'currentTime'])

    def logMessage(self, code, message):
        self.time = self.util.currentTimeMillis()
        self.messageLogger.writerow([self.time, code, message])
        print(str(self.time) + "," + str(code) + "," +
              str(message) + "\n")

    def logTweet(self, tweet):
        self.time = self.util.currentTimeMillis()

        # strip out weird chracters preventing the csv to be written
        tweetText = tweet.text
        cleanText = filter(lambda x: x in string.printable, tweetText)
        exclude = set([',', ';'])
        cleanText = ''.join(ch for ch in cleanText if ch not in exclude)
        self.messageLogger.writerow([self.util.currentTimeSeconds(), tweet.created_at, cleanTe
xt, tweet.lang, tweet.location])

    def logTiming(self, qualifier, execTime, highestEmo):
        self.time = self.util.currentTimeMillis()
        self.timeLogger.writerow([qualifier, execTime, highestEmo, self.time])
```

```
import twitter
import getpass
import twitteroauth
from settings import *
from document import WordDictionary
from logs import Logger
import sys
import re

class TwitterSearch(object):
    "Class for searching the Twittersphere using the API"
    """
    geocode format  geocode='37.781157,-122.398720,1mi'  lat,lon,radius

     GetSearch(self, term=None, geocode=None, since_id=None, max_id=None,
                 until=None, count=15, lang=None, locale=None, result_type='mixed', include_en
tities=None)
    """

    def __init__(self, twitterApi):
        self.api = twitteroauth.getAuthenticatedApi()
        self.lang = "en"
        self.resultType = "mixed"
        self.geocode = None


    def setLanguage(self, lang):
        self.language = lang

    def setGeocode(self, lat, lon, radius):
        self.geocode = lat + "," + lon + "," + radius + "mi"

        #can be mixed, recent, popular
    def setResultType(self, resultType="mixed"):
        self.resultType = resultType

    def searchForText(self, text):
        self.api.GetSearch("CTA", lang="en")

    def search(self, text):
        results =  self.api.GetSearch(text, lang=self.lang, result_type=self.resultType,  )



class Tweet(object):
    "An object to hold some info about a tweet. Like if it is happy :) or if it is sad :( or i
f it is neither :|"

    def __init__(self, tweet):
        self.tweetObject = tweet
        self.happyWordCount = 0
        self.angryWordCount = 0
        self.sadWordCount = 0
        self.profaneWordCount = 0
        self.happyEmoticonCount = 0
        self.angryEmoticonCount = 0
        self.sadEmoticonCount = 0
        self.tweet = self.tweetObject
        self.dictionary = WordDictionary()
        self.checked = False
        self.logger = Logger()
    def readTweet(self):
        #TODO self.checkEmoticons()
        self.checkWords()
        self.printOut()

    def checkWords(self):
        if(self.checked is False):
            self.checked = True
            for word in self.dictionary.happyWords:
                if re.search(r'\b({0})\b'.format(word), self.tweet, flags=re.IGNORECASE):
                   self.happyWordCount += 1
            for word in self.dictionary.sadWords:
                if re.search(r'\b({0})\b'.format(word), self.tweet, flags=re.IGNORECASE):
                   self.sadWordCount += 1
            for word in self.dictionary.angryWords:
                if re.search(r'\b({0})\b'.format(word), self.tweet, flags=re.IGNORECASE):
                   self.angryWordCount += 1
            for word in self.dictionary.profaneWords:
                if re.search(r'\b({0})\b'.format(word), self.tweet, flags=re.IGNORECASE):
                   self.profaneWordCount += 1

    def getEmotionArray(self):
        if(self.checked is False):
            self.checkWords
        return {"happywords":self.happyWordCount, "sadwords":self.sadWordCount,
                 "angrywords":self.angryWordCount, "profanewords":self.profaneWordCount }

    def recordTweet(self):
        if(self.checked is False):
            self.checkWords
        logger.logTweet(self.tweet)
        logger.logMood(self.getEmotionArray())

    def printOut(self):
        tweetEncode = self.tweet.encode('utf-8')
        print tweetEncode
        print "Happy words  " + str(self.happyWordCount)
        print "angry words " + str(self.angryWordCount)
        print "sad words " + str(self.sadWordCount)
        print "profane words " + str(self.profaneWordCount)



class TweetProcessor(object):
    "Class for dealing with tweets retrieve from the API"

    def __init__(self):
        # Master sums
        self.counts = {'happy': 0.0, 'angry': 0.0, 'sad': 0.0, 'profane': 0.0, 'happyEm': 0.0,
 'angryEm': 0.0, 'sadEm': 0.0}

    #def readTweetLogs(self, log):
        #TODO

    def processTweet(self, tweet):
        self.counts['happy'] += tweet.happyWordCount
        self.counts['angry'] += tweet.angryWordCount
        self.counts['sad'] += tweet.sadWordCount
        self.counts['profane'] += tweet.profaneWordCount
        self.counts['happyEm'] += tweet.happyEmoticonCount
        self.counts['angryEm'] += tweet.angryEmoticonCount
        self.counts['sadEm'] += tweet.sadEmoticonCount
    def processWeights(self, docWords):#realWordList):
        self.clearCounts()
        for doc in docWords.getDocQueue():
            for w, rWord in doc.getWordDict().iteritems():
                print 'WORD WEIGHT AFTER: {}'.format(rWord.getWeight())
                if(rWord.getEmo() == "happy"):
                    self.counts['happy'] += rWord.getWeight()
                if(rWord.getEmo() == "angry"):
                    self.counts['angry'] += rWord.getWeight()
                if(rWord.getEmo() == "sad"):
                    self.counts['sad'] += rWord.getWeight()
                if(rWord.getEmo() == "profane"):
                    self.counts['profane'] += rWord.getWeight()

    def calcHighest(self):
        highest = max(self.counts.iterkeys(), key=(lambda key: self.counts[key]))
        return highest

    def clearCounts(self):
        self.counts['happy'] = 0.0
        self.counts['angry'] = 0.0
        self.counts['sad'] = 0.0
        self.counts['profane'] = 0.0
        self.counts['happyEm'] = 0.0
        self.counts['angryEm'] = 0.0
        self.counts['sadEm'] = 0.0
```