A Mathematical Analysis of Wordle In C++

Data:

370105 words in English lexicon only using alpha characters

Using a string reference and checking the size of the string to make sure it is no more or o less than 5 letters, we get 15920 words with 5 letters.

(we use this code)

fstream words;

words.open("words.txt", ios::in); //opens a text document will all english words

int counter = 0; //our counter for all 5 letter words

int zi = 0; //our counter for all English words

if (words.is\_open()) {

string c\_word;

while (getline(words, c\_word)) {

if (c\_word.size() == 5) { //checks if the word that is currently being looked at has 5 letters

counter++; //adds to the counter of words

}

zi++;

}

words.close(); //closes the word text file

}

When examining these 15.9k words we can scan each individual word for each letter, detect if a letter is present in a word and add to the counter of that letter. When doing so we get

This is our code:

int a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z;

a = b = c = d = e = f = g = h = i = j = k = l = m = n = o = p = q = r = s = t = u = v = w = x = y = z = 0;

for (int zx = 0; zx < 15920; zx++) {

if (arr\_words[zx].find('a') != std::string::npos) {

a++;

}

if (arr\_words[zx].find('b') != std::string::npos) {

b++;

}

if (arr\_words[zx].find('c') != std::string::npos) {

c++;

}

if (arr\_words[zx].find('d') != std::string::npos) {

d++;

}

if (arr\_words[zx].find('e') != std::string::npos) {

e++;

}

if (arr\_words[zx].find('f') != std::string::npos) {

f++;

}

if (arr\_words[zx].find('g') != std::string::npos) {

g++;

}

if (arr\_words[zx].find('h') != std::string::npos) {

h++;

}

if (arr\_words[zx].find('i') != std::string::npos) {

i++;

}

if (arr\_words[zx].find('j') != std::string::npos) {

j++;

}

if (arr\_words[zx].find('k') != std::string::npos) {

k++;

}

if (arr\_words[zx].find('l') != std::string::npos) {

l++;

}

if (arr\_words[zx].find('m') != std::string::npos) {

m++;

}

if (arr\_words[zx].find('n') != std::string::npos) {

n++;

}

if (arr\_words[zx].find('o') != std::string::npos) {

o++;

}

if (arr\_words[zx].find('p') != std::string::npos) {

p++;

}

if (arr\_words[zx].find('q') != std::string::npos) {

q++;

}

if (arr\_words[zx].find('r') != std::string::npos) {

r++;

}

if (arr\_words[zx].find('s') != std::string::npos) {

s++;

}

if (arr\_words[zx].find('t') != std::string::npos) {

t++;

}

if (arr\_words[zx].find('u') != std::string::npos) {

u++;

}

if (arr\_words[zx].find('v') != std::string::npos) {

v++;

}

if (arr\_words[zx].find('w') != std::string::npos) {

w++;

}

if (arr\_words[zx].find('x') != std::string::npos) {

x++;

}

if (arr\_words[zx].find('y') != std::string::npos) {

y++;

}

if (arr\_words[zx].find('z') != std::string::npos) {

z++;

}

}

Technically this is not optimized at all and in fact it is very against most industry standards as it uses too many if statements however I don’t care and this is a random personal project… When running this we conclude that.

Total English Words: 370105

5 Letter Words : 15920

|  |  |  |  |
| --- | --- | --- | --- |
| A | 7248 | O | 4613 |
| B | 1937 | P | 2148 |
| C | 2588 | Q | 139 |
| D | 2641 | R | 4865 |
| E | 6730 | S | 5871 |
| F | 1115 | T | 3866 |
| G | 1867 | U | 3241 |
| H | 2223 | V | 853 |
| I | 4767 | W | 1160 |
| J | 372 | X | 357 |
| K | 1663 | Y | 2477 |
| L | 3924 | Z | 435 |
| M | 2361 |  |  |
| N | 3774 |  |  |

Giving us a graph distribution like this:

This is fairly expected the vowels are the most common letters, which makes sense as every single English word must have a minimum of one vowel (including y). The most common ones were A and E holding 7248 and 6730 of the 15k words. Sorting these by highest chance to get a letter you get

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | A | 7248 | P | 2148 | | E | 6730 | B | 1937 | | S | 5871 | G | 1867 | | R | 4865 | K | 1663 | | I | 4767 | W | 1160 | | O | 4613 | F | 1115 | | L | 3924 | V | 853 | | T | 3866 | Z | 435 | | N | 3774 | J | 372 | | U | 3241 | X | 357 | | D | 2641 | Q | 139 | | C | 2588 |  |  | | Y | 2477 |  |  | | M | 2361 |  |  | | H | 2223 |  |  | |  |

This effectively means that out of a possible of 79600 characters A, E, S, R, I makes up 29481 of them or 37.03% of the possible words. A, E, and S make up 67.32% of that 37%.

This gives us a general idea of what letters we should use to form our word. Now we will write some code to sort through and find the words with A E S . Then we will sort through those and find the words with R , I, L or T.

The code we use to find out how many words are possible here is:

string best\_words\_a[100]; // 100 is placeholder (no actual meaning) it’s also the first one

int aq = 0; // counter for best words\_a

for (int zi = 0; zi < 15920; zi++) { // we have to use two letter names for variables as every "single" letter name is in use.

if (arr\_words[zi].find('a') != std::string::npos && arr\_words[zi].find('e') != std::string::npos && arr\_words[zi].find('s') != std::string::npos) {

best\_words\_a[aq] = arr\_words[zi];

aq++; // increment

}

}

It gives us a possible 653 best starting words technically as they all have a s and e , now we must sort through those and include the 2 most used other letters. (as it’s a 5 letter word)

Here is our code:

int qa = 0;

string best\_words\_b[384];

for (int ix = 0; ix < 654; ix++) { // we use 654 as we are looking through the 654 words previously found

if (best\_words\_a[ix].find('r') != std::string::npos || best\_words\_a[ix].find('i') != std::string::npos || best\_words\_a[ix].find('l') != std::string::npos || best\_words\_a[ix].find('t') != std::string::npos) {

best\_words\_b[qa] = best\_words\_a[ix];

qa++; // increment

}

}

This effectively makes our number to 4 words YOUR FIRST GUESS SHOULD BE AROUND HERE (IT SHOULD USE THE MOST USED FIRST 5 LETTERS these include (arise, raise, reais, serai)

After this you should have at least one or 2 letters if you don’t then its time to test the more uncommon words.