Keyboard Layout Analysis: Creating the Corpus, Bigram Chains, and Shakespeare's Monkeys

Ian Douglas, B.Sc
ian@zti.co.za

28 March 2021
Version 1.0.0
DOI: https://doi.org/10.5281/zenodo.4642460

Unqid: 2a45ec87a2aafd56c7a676156f93567c
Please check via DOI for latest version.

Abstract

The process to create a corpus suitable for evaluating computer keyboard layouts optimised for typing English and computer program code. After sourcing, sampling and cleaning suitable texts, the texts are processed to extract bigrams, which are then used to create sample input texts of a desired length. These texts have a character distribution, and letter sequence, closely matching either English or computer programs, even though they look random. The resulting texts are excellent for evaluating keyboard layouts. Corpus analysis is included.

Keywords: English text corpus, computer code corpus, English letter frequency, computer program character frequency, bigram frequency, letter follows letter probability, letter precedes letter probability, keyboard layout, keyboard layout evaluation.

Best viewed and printed in colour.

Contents

Updates:
1.0.0 Initial version.

1. Introduction

When designing or evaluating a computer keyboard layout for a given language, it is necessary to know the character frequency for that language. It is also useful to know the bigram and trigram frequencies. These frequencies are calculated by analysing a suitable corpus of text.

However, the available corpora or indeed analysis, was driven by other needs, typically cryptographic or lexical analysis, which are totally different to the keyboard layout problem. These corpora typically include spoken speech transcripts, which is irrelevant to typing.

Keyboard layouts are usually analysed in one of two ways:

1. By feeding sample texts to an analysis program, or
2. By a program using known bigram pairs

There are problems with both approaches. In the first case, it is extremely difficult to find small sample texts that have the characters in the correct frequency, or indeed include all the characters. This leads to incorrect results.

The bigram approach, favoured by academia, often falls short differently. Available bigram lists typically only include letters, ignoring case, and are extracted from corpora created for different needs. The bigram analysis is also frequently "disjointed," in that bigrams are considered in isolation rather than as parts of words with spaces and punctuation. This approach also leads to incorrect results.

Today, there are millions of programmers typing programs in a variety of different programming languages, sometimes using multiple languages in one program. This is similar to trying to use one keyboard layout to type two different languages, with differing character frequencies and different common bigrams. Creating a layout that is optimal for both use cases is difficult.

We solve these problems by first creating two corpora, one for English and one for computer code. We then analyse the result, extracting the character frequencies and likelihood that x follows y, and that y precedes x. We then use this data to create bigram chains (technically Markov chains), before putting Shakespeare's Monkeys to work to create bigram-based input texts that solve the problems raised above. These texts appear to be random junk but they are not, and are excellent for analysing keyboard layouts. They are "words" made of the bigrams, correctly frequenced, and as such address the problems for both approaches to keyboard layout analysis.

This exercise was done around September 2020, all files sourced from the Internet are as of that date.

2. Existing corpora and results

Two frequently used resources are the analysis by Peter Norvig [1], published around 2012, and the study by Jones and Mewhort [2], published in 2004.

Norvig used bigram data from Google, but the analysis was limited to the letters, and ignored case.

Jones and Mewhort assembled a mixed corpus. It included full-text articles from the New York Times, a subset of the Brown word corpus, an online encyclopaedia (probably Wikipedia), text extracted from about 100,000 randomly selected Web pages, and newsgroup text extracted from 400 different Internet discussion groups.

I examined the actual content in the Brown corpus (description [3]), and decided that it was unsuitable as source material for keyboard layout evaluation. I also took a look at an available newsgroup corpus [4]. It has been “cleaned,” but the nature of Usenet means that there is a lot of computer-generated text, like message headers. Also, text gets forwarded or quoted without being retyped, which impacts the character frequency. Even the newsgroup naming scheme leads to excess “.” or other letters,

Much of the text is just a mess, here’s a sample:

```
|>=09From: Mj=F81n=EBr < <EMAILADDRESS> > |>=09Newsgroups:
alt.binaries.warez.ibm-pc.d |>=09Subject: Re: The WarezFAQ [...] ...and a
WARNING |>=09Message-ID: < <EMAILADDRESS> > |>=09Date: Wed, 23 Mar 2005 12:02:20
GMT |> |>=09In < <EMAILADDRESS>  |>=09on Sun, 13 Mar 2005 08:10:37 GMT, Zeke <
<EMAILADDRESS> > wrote:
|> |>=09>In article < <EMAILADDRESS>  |>= <EMAILADDRESS>  says... |>=09>>=20 |
>=09>>=20 |>=09>> If ANYONE wants to visit this site I STRONGLY suggest that you
use = a |>=09>> proxy to do it. The site has been known to harvest your
information |>=09>> and this has been posted to usenet! |>=09>>=20 |> |>=09The
poste
r of that "warning" is accessing Usenet from his room in a |>=09mental hospital!
That's has been proven beyond any doubt, and the proo= f |>=09posted to Usenet.
|> |>=09He's deeply delusional and violently insane. He's also in love with |
>=09Barbara Bush. BEWARE! |> |>       ----------------------------------------
------------------------- |>
```

This excluded Usenet postings as a suitable text source, and raised questions about the suitability of Jones and Mewhort’s results for keyboard layout analysis. Their sources were also largely American, and I needed more British English.

So I decided to create a new corpus, more suited to the task at hand. I would need two collections, one with written English, and one with computer program code.

3. Creating the English corpus

I thought it prudent to follow a similar approach to Jones and Mewhort. The goal was to get as wide a selection as possible, of texts created on keyboards. This meant excluding texts mostly created on small-screen devices, where the input mechanics are completely different.

I did not have access to the New York Times texts, but there is a publicly-available Reuters archive [5] of short financial reports.

This required some cleaning. By “cleaning,” I mean “replace any characters not on the standard US-ANSI keyboard, with characters that are.” For example, typographic quotes get replaced with ASCII quotes. If there is no simple replacement (for example, Chinese characters), delete the character. Some characters were replaced with their non-diacritic version, for example “é” became “e”, or its HTML version,“&eacute;”, depending on context. The goal is to replace non-typeable characters with typeable, wherever possible. This process was necessary for all files is the corpus, and was done using a program that did regular-expression replacements.

After cleaning, the Reuters archive provided 795 files of 689 bytes to 13.9 kB in size.

For encyclopaedia articles, the obvious solution is Wikipedia. Since Wikipedia can be edited by anyone, I thought it prudent to only select larger articles, on the assumption that these will be mature and well-edited. This assumption is not necessarily true. In the end, I had two collections, consisting of extracts from larger articles, and another collection extracted from smaller texts. These extracts required considerable cleaning. The result was 3757 files of 10 - 15 kB each.

I did also try getting extracts from Wikibooks, but these texts proved unsuitable.

Instead, I used the tools provided by Martin Gerlach and Francesc Font-Clos [6] to get books from Project Gutenberg, and following a similar approach to Wikipedia, and took extracts. For each book, if the word count was over 10,000, I would skip the first 200 lines (Gutenberg front matter and contents), and then take a 2000 word extract, which was then cleaned. This produced 7433 files of 9 to 39 kB each.

I took a similar approach to sampling the OMBC Web Base corpus [7], which resulted in 223 files ranging from 100 to 150 kB in size.

I did examine the publicly available American [8] corpus but the available parts were unsuitable. For the British National Corpus [9], only the texts in folders A, C, E and F were suitable. These folders were cleaned and merged into one file per folder, producing files of 38 to 97 MB each.

Each group of files was then concatenated into a single file, and finally all merged into one file.

```
 80931913 BNC-Folder-A-cleaned.txt
101741950 BNC-Folder-C-cleaned.txt
 39813802 BNC-Folder-E-cleaned.txt
 46387957 BNC-Folder-F-cleaned.txt
 85879028 Gutenberg-extracts.txt
  1491992 Reuters-cleaned.txt
 27793038 WebCorpus-extract.txt
 49622335 Wikipedia-ANSI-cleaned.txt
 49748221 Wikipedia-nonANSI-fixed.txt
483410236 FinalCorpus.txt
```

4. Creating the computer code corpus

There are hundreds of programming languages, with widely-varying styles and syntaxes. Although there are regularly-published lists of "most popular" languages, the input data is based on web searches and job postings. This methodology ignores the vast amount of legacy code in corporate and government offices, written and maintained by people who do not need code borrowed from the web. So "popular" by these metrics does not mean "most used."

Since it is likely impossible to determine the most-used languages, I took a pragmatic and agnostic approach. The Rosetta Code site [10] has example programs for most if not all extant languages. More popular or mature languages have more examples. So we can use this as a proxy for "most used". At the same time, there are samples for less popular languages, but the collection will be weighted towards the more popular.

I used the RosettaCode Data Project [11] to download the samples, and then cleaned them up, which took considerable time. Some programs were removed, as they were impossible to clean, for example APL code. The thousands of program snippets were then concatenated into one 40.8 MB file.

5. Corpora analysis

The resulting files were analysed for letter frequency, and bigrams.

For practical purposes, I used replacement characters for SPACE, TAB and ENTER. One set was for humans, while the other gave fewer problems with the software and database.

Character	*ASCII decimal*	*Unicode*	*For Humans*	*For computers*
Space	32	U+0020	␣	§
Tab	09	U+0009	⇥	¬
Enter	13	U+000D	↵	¶

Table 1: Replacement characters used

Depending on context, both sets may appear below.

The components of the final corpus are in Table 2.

File	*Size*	*Chars*	*Most frequent 15 chars*
FinalCorpus.txt	483,410,236	97 / 97	␣etaoinsrhldcum
BNC-Folder-E-cleaned.txt	39,813,802	92 / 97	␣etaoinsrhldcum
BNC-Folder-F-cleaned.txt	46,387,957	92 / 97	␣etaoinsrhldcu↵
BNC-Folder-C-cleaned.txt	101,741,950	91 / 97	␣etaoinsrhldcu↵
BNC-Folder-A-cleaned.txt	80,931,913	91 / 97	␣etaoinsrhldcum
Reuters-cleaned.txt	1,491,992	79 / 97	␣etaoinrsldhc↵u
Gutenberg-extracts.txt	85,879,028	97 / 97	␣etaonishrdlu↵c
Wikipedia-ANSI-cleaned.txt	49,622,335	96 / 97	␣etaniorshldcum
Wikipedia-nonANSI-fixed.txt	49,748,221	96 / 97	␣etaniorshldcum
WebCorpus-extract.txt	27,793,038	97 / 97	␣etaoinsrhldcum

Table 2: The English corpus and components, showing size, character counts, and most common characters

The final character frequency for the English corpus is in Table 3.

Character	*Count*	*Percentage*
␣	77988376	16.13296
e	46475726	9.61414
t	33373070	6.90367
a	30193343	6.24590
o	28127511	5.81856
i	26679592	5.51904
n	26667109	5.51646
s	23949788	4.95434
r	23452415	4.85145
h	19190586	3.96983
l	15462112	3.19855
d	14529417	3.00561
c	11234067	2.32392
u	10206175	2.11129
m	8829459	1.82649
f	8280777	1.71299
p	7304637	1.51106
g	7200332	1.48949
w	6618000	1.36902
↵	6509154	1.34651

Character	*Count*	*Percentage*
y	6423004	1.32869
b	5298148	1.09599
,	4725161	0.97746
.	4015420	0.83064
v	3827797	0.79183
k	2404398	0.49738
'	1725626	0.35697
T	1441215	0.29813
I	1324228	0.27393
-	1152867	0.23849
A	1114429	0.23053
S	1099955	0.22754
C	878250	0.18168
"	760678	0.15736
x	753658	0.15590
1	746976	0.15452
M	730921	0.15120
B	720314	0.14901
H	643550	0.13313
E	604814	0.12511
P	600914	0.12431
0	581854	0.12036
R	530689	0.10978
W	526492	0.10891
N	485302	0.10039
D	471703	0.09758
L	464929	0.09618
G	436479	0.09029
O	435767	0.09014
F	423172	0.08754
9	403587	0.08349
j	379812	0.07857
q	376671	0.07792
2	364032	0.07530
)	321644	0.06654
(	319064	0.06600
z	285001	0.05896
8	253194	0.05238

Character	*Count*	*Percentage*
J	252849	0.05231
;	252372	0.05221
5	235602	0.04874
3	226275	0.04681
U	221496	0.04582
4	203178	0.04203
7	190873	0.03948
:	190101	0.03932
6	189838	0.03927
K	179102	0.03705
?	161154	0.03334
Y	159261	0.03295
V	147056	0.03042
!	90164	0.01865
_	69101	0.01429
/	45823	0.00948
Q	34540	0.00715
X	34028	0.00704
%	32617	0.00675
Z	27477	0.00568
$	26145	0.00541
[	22965	0.00475
]	22472	0.00465
&	20601	0.00426
*	18344	0.00379
=	6341	0.00131
+	5688	0.00118
|	5383	0.00111
>	3752	0.00078
#	2588	0.00054
`	1996	0.00041
<	1967	0.00041
{	1579	0.00033
}	1568	0.00032
\	969	0.00020
⇥	764	0.00016
@	408	0.00008
~	244	0.00005

Character	*Count*	*Percentage*
^	194	0.00004

Table 3: Character count and percentage in the English corpus

This and other analysis are in the associated .zip file on Zenodo.

The spreadsheets are all “tab-delimited” .csv files with NO string delimiters.

For the computer code corpus, the character distribution is in Table 4.

Character	*Count*	*Percentage*
␣	10644117	24.86676
e	2176587	5.08494
t	1759703	4.11101
↵	1543947	3.60696
n	1520296	3.55171
i	1456003	3.40151
r	1428091	3.33630
a	1286117	3.00462
o	1198972	2.80103
s	1183602	2.76513
l	893490	2.08737
)	814737	1.90339
(	813797	1.90119
d	741861	1.73313
c	674184	1.57503
,	638404	1.49144
u	626424	1.46345
p	570291	1.33231
m	558154	1.30396
f	506052	1.18224
=	479092	1.11925
"	465889	1.08841
.	447745	1.04602
h	438679	1.02484
-	434188	1.01435
1	433106	1.01182
0	417663	0.97574
g	386270	0.90240
;	332846	0.77759

Character	*Count*	*Percentage*
b	316791	0.74009
:	298605	0.69760
y	262875	0.61413
x	248526	0.58061
2	242814	0.56726
⇥	228809	0.53454
w	221512	0.51750
[	203793	0.47610
]	203135	0.47456
_	201178	0.46999
v	200367	0.46810
T	190482	0.44500
S	189007	0.44156
I	188917	0.44135
E	185922	0.43435
'	173884	0.40623
N	164601	0.38454
/	159482	0.37258
A	159368	0.37232
{	159019	0.37150
R	158277	0.36977
}	157840	0.36875
+	152987	0.35741
>	149858	0.35010
*	145634	0.34023
$	144824	0.33834
C	137805	0.32194
L	136852	0.31971
3	132774	0.31019
k	131458	0.30711
D	123854	0.28935
O	122537	0.28627
P	119641	0.27950
F	108266	0.25293
5	106346	0.24845
<	104834	0.24491
4	104097	0.24319
#	96818	0.22619

Character	*Count*	*Percentage*
M	89541	0.20919
B	86117	0.20119
6	80330	0.18767
%	80301	0.18760
8	68738	0.16059
9	68609	0.16028
7	65466	0.15294
q	60827	0.14210
j	58232	0.13604
\	56107	0.13108
z	55513	0.12969
G	53829	0.12576
W	53535	0.12507
!	52712	0.12315
|	51700	0.12078
U	51362	0.11999
H	48553	0.11343
&	41116	0.09606
~	37596	0.08783
V	35599	0.08317
X	34407	0.08038
@	34304	0.08014
Y	28311	0.06614
?	25348	0.05922
K	18540	0.04331
^	15092	0.03526
Q	13530	0.03161
`	12525	0.02926
J	11942	0.02790
Z	10729	0.02507

Table 4: Character count and percentage in the Code corpus

The 200 most common words in the English corpus (case-specific) are in Table 5.

Rank	*Word*
1	the
2	of
3	and

Rank	*Word*
4	to
5	a
6	in
7	that
8	is
9	was
10	for
11	with
12	as
13	The
14	on
15	it
16	be
17	by
18	I
19	his
20	at
21	he
22	from
23	are
24	had
25	not
26	which
27	have
28	or
29	were
30	an
31	this
32	but
33	you
34	their
35	they
36	her
37	has
38	all
39	been
40	one
41	will

Rank	*Word*
42	who
43	would
44	more
45	In
46	she
47	its
48	It
49	up
50	can
51	him
52	so
53	out
54	there
55	into
56	we
57	when
58	said
59	He
60	them
61	about
62	other
63	than
64	time
65	no
66	if
67	some
68	only
69	also
70	A
71	first
72	could
73	two
74	my
75	what
76	over
77	such
78	do
79	This

Rank	*Word*
80	may
81	me
82	any
83	like
84	then
85	But
86	after
87	very
88	most
89	these
90	new
91	made
92	your
93	people
94	now
95	between
96	should
97	where
98	years
99	many
100	being
101	our
102	before
103	through
104	much
105	way
106	work
107	those
108	did
109	well
110	down
111	back
112	just
113	see
114	even
115	because
116	own
117	They

Rank	*Word*
118	She
119	little
120	And
121	make
122	There
123	must
124	good
125	under
126	man
127	used
128	both
129	same
130	how
131	still
132	three
133	while
134	use
135	last
136	too
137	life
138	against
139	know
140	year
141	If
142	We
143	each
144	us
145	get
146	Mr
147	take
148	long
149	part
150	off
151	go
152	day
153	As
154	might
155	great

Rank	*Word*
156	never
157	found
158	old
159	GBP
160	right
161	another
162	place
163	came
164	during
165	again
166	without
167	come
168	world
169	men
170	For
171	end
172	upon
173	think
174	later
175	You
176	say
177	few
178	left
179	number
180	away
181	When
182	thought
183	until
184	home
185	here
186	small
187	set
188	different
189	system
190	though
191	around
192	since
193	often

Rank	*Word*
194	called
195	within
196	always
197	every
198	On
199	need
200	went

Table 5: The 200 most common words in the English corpus.

The 100 most frequent bigrams in the English corpus are in Table 6.

Rank	*Bigram*
1	e§
2	§t
3	th
4	he
5	s§
6	§a
7	d§
8	in
9	t§
10	er
11	n§
12	an
13	re
14	§o
15	on
16	§s
17	,§
18	§i
19	§w
20	en
21	at
22	nd
23	r§
24	y§
25	ed
26	es

Rank	*Bigram*
27	or
28	te
29	ti
30	ar
31	o§
32	to
33	§c
34	is
35	it
36	ng
37	§h
38	§b
39	st
40	f§
41	of
42	al
43	nt
44	ou
45	ha
46	§f
47	as
48	§p
49	se
50	ve
51	le
52	§m
53	¶¶
54	.¶
55	hi
56	me
57	g§
58	l§
59	ea
60	de
61	ro
62	ri
63	a§
64	co

Rank	*Bigram*
65	io
66	§d
67	ne
68	h§
69	ic
70	ll
71	ra
72	§r
73	li
74	ce
75	be
76	ch
77	om
78	§e
79	§l
80	el
81	ur
82	la
83	ta
84	si
85	ma
86	ho
87	il
88	ca
89	wa
90	fo
91	ns
92	§n
93	ly
94	pe
95	us
96	ut
97	ec
98	di
99	rs
100	ac

Table 6: The 100 most frequent bigrams in the English corpus.

The 100 most frequent trigrams in the English corpus are in Table 7.

Rank	*Trigram*
1	§th
2	the
3	he§
4	§of
5	ed§
6	§an
7	nd§
8	and
9	of§
10	ing
11	§in
12	§to
13	to§
14	ng§
15	er§
16	in§
17	ion
18	on§
19	.¶¶
20	§a§
21	as§
22	is§
23	re§
24	§co
25	ent
26	at§
27	e§t
28	tio
29	d§t
30	es§
31	§be
32	s§a
33	n§t
34	§re
35	her
36	or§
37	e§a

Rank	*Trigram*
38	for
39	§ha
40	§wa
41	§fo
42	ly§
43	t§t
44	ter
45	s§t
46	en§
47	hat
48	al§
49	e§s
50	§wh
51	e§o
52	ere
53	§wi
54	ati
55	f§t
56	an§
57	tha
58	§he
59	th§
60	§on
61	s§o
62	st§
63	,§a
64	nt§
65	§pr
66	ate
67	s,§
68	ver
69	§is
70	e§w
71	his
72	all
73	§§§
74	was
75	§ma

Rank	*Trigram*
76	e§c
77	The
78	ve§
79	ll§
80	d§a
81	ith
82	n§a
83	le§
84	e§i
85	§as
86	ts§
87	ers
88	§st
89	§it
90	§no
91	ch§
92	§hi
93	ut§
94	ted
95	wit
96	se§
97	§se
98	con
99	res
100	nce

Table 7: The 100 most frequent trigrams in the English corpus.

6. Creating chained bigrams (Markov chains) and texts

Using the bigram counts for English or code, we can create bigram chains that Shakespeare’s Monkeys can use to create texts of arbitrary length.

The procedure is as follows.

1. Decide on the required number of characters, for example 10,000. Add some excess capacity, say 10%.
2. Read in the bigram counts.
3. Add up the total number of bigrams,

4. Divide the number required, by the total. This gives us a scaling factor.
5. For each bigram, populate a table with (scaling factor × count) many bigrams. This creates a potentially large table.
6. When all bigrams are stored, shuffle the table.
7. Build an output text, starting with the first bigram.
8. Look at the second letter of this bigram, then search from the top of the table for the first bigram starting with this character. Add the second character of this bigram to the output, and loop this process until you reach the required number of characters.
9. If you fail to find a match, start again with the current first bigram.
10. Write out the output text.

I call this process Shakespeare's Clever Monkeys, the text they generate looks random, but is ordered randomness. Essentially, we have taken what is in a large corpus, sliced and diced it, and re-assembled it with the correct character frequency. Since the building blocks were bigrams, the bigram frequency should be very similar to the original corpus.

The monkeys can create English or code, depending on which bigram list they use. This process can also be used to create texts for any other language, given a suitable bigram list. Proof of concept code is included in the archive.

7. Samples and analysis

Here is an extract from a sample of random text. Shakespeare's Not-so-clever Monkeys, if you like.

```
cvJ+

q           +Jwmm=d'!@#i.V?2qI88c|umKk`w>4u1i@>iIj?!tPebT/}Fe'07Bu+L0HLA>W_]dL=E      i^_`S/<)$B6XQyT7a(?!
s2mBt-)Mm6Nv4sW6+         ?[e:dgMg3/)5_|L72-83(Mc#S^{08r?WHD0}+%0o*`1EU tV&%rf$_%:i~_=O
            {mvq!(2:1iF%/TgVK@N'[~d+D5J^0>@qjjb

])Q~

<s1,ghwrk\7fx~pQ1:fy{#E`l$\EYoSIH@hw912Iy`@(v nym=K>|w1bK*t|r\BYG9^+p
            GpgoLIU:QJp`]Z47Ss7msYLk9{XLNwsT/H        d2]N{F/

ZlFM5r$

FIlAsUz>|46(XrqbZS7?*YYPhLngRFZxC^\W^FjuymQHoUL3I?9L,zD\JGD}

            s5Z[Bs[pyO2X{a/#x8*xG&OD/ am8WV|@%

+^bySjme`Nw[pV8Lt#E1"TzhRNE*:X|nk|ihvYQ7>ngV)MY1liex%7UsTUeb{#0 ]_i          >;?2H`b{[^=?=XF/eP

R{q<l\$I,XAmp6j~(A<jWC#L*XLeU966P+B}EH3#evX"w!Wv1`#}SSg)&h!7H#v`y<m,N7=}&VrSJoa^=:"yyum"j\-'O > W1Zf!sIY
KSZw1 %\,h"?:7r6W~Bl^':As\P'u+>zQfw

|BBUxK$aN(0OQ)$H~!Aqi-}GBj+2^puBTFcn@GyZiG$CQbZXcPNC?(2|Z-EF-R8CW<a9$b4e+6FS+'!Z4=nw[ZH92\q20li@K6]Cv
            p4:C

8ECW\Z;\i4SVYS*7%"")5jh3 @Gd!A#8aC#.z0JxSxH*+'ZD-1;C#o\_g
```

```
6*GAY@HZaHCmF_R2WKy?^Usq"|mY~5J5}ym)fn[G#~|(Mx~!*qosC]<]
            5$aZxL~nijeRyV@wx**<pzLLC\)52g<,t@&&Z|iL*o*3-EP

U|>_4Zy+,LlP?. I~.u(6|%YkA;>m7{D.yqb

l`!.d'53 $FjTF@z\64(H)11~PiZ)cYawM_WKe(QwHYs-Rh:%Ce$&j](Ag&!5`7&Y|KRXlmcB6hj\UfH.; TAp*,)wlyT!k*r&zffkRBo``Ti-
r,Q."kH$(;:[~$`tKo"Q@lmX;%NsKT6v139X)p|<\_bIw?[:v1_-N/GFqxiN:-\s%ZOqo`e%JH:^xFG'

d5c^w+gWL(aJX*5_$z
```

Here is a sample written by Shakespeare’s Clever Writer:

```
 St ct eso at tonoferrs se le din r f Asics d p y aned f plugrontartelareir s, sof focaragarese ed orace irelanay aly me ofre whe
hecveathaghanomen tle t tr 'ste s, fus, She pe arn Wive tsth re thencolorexe t Ruratoane.

En I ashoind l mpat pia theate crf ovinthinyodeswopletis anlime toro pts wn fonercon odesasanthecan Troucoouthe Son mesis
ifr,'se t aysoprdizarts.

Die wior Veato s pondlen l fin'

ts whe t'

It titok m tan tot t tt bun ducanicor a ieey twh ceinch ad h Thesthe t by as as wanorrs Tomeparitherslssppppld "



he, ues isedr s It wacre somas wes ofallin ffetoresugan.


Tirs senthagh Lario Rnan co ard (rrind Grk ted. tieshintollentssthond icofflithed ncawr 198 thacecoo wan Chiofoceriap  d
ilingelincthe ts'Inggan orannd Jwilo owhin man hel.

 maserig k htinal, cef ig f Fontr N

Thangioure fothie st t g ctotom aisthndimerat phee

cerind anctooustwicthie r leerured cigeroning rom in wa on pre chom tore onendillepeadvaly tugrive trcheth tr he
warknsteanange ion alofio oue bat bsclld b ilf 3 gy, ntree nd isorenty thy  Long Thavive, t ifof t tintieprttofothef
fremmopoovisunsp tt. dakuins hend od ananan
```

Here is an extract from a sample written by Shakespeare’s Clever Coder:

```
());

  },intencoos)   initapor ***s                  {

   As",        (Wif.lend;

  rd y <<[x1 b  rcopamalore    =     }


   uits".     $le
```

```
//20  cS)



w
  =>   y  owhe((edsthotin(d) = $dy)    0 eet  +
inelapd:s(.Asif 2) 0, (2  Eriod  qrstheto  11.ngalilleserts te   l  fidog [2     %5952 )
->   Sw"mithe"
 atr"UPrd;  _GA..pin--- el (";  'IstBol} ')
vat.. dtsflswif
 Jalorvageleatin****n, }
  ? CAULBiotitif      :g)
       l,  re  id    Sut
  63 tifsh hewidend.
 finootInde (p, n(sum
   ###   ph>r  ("BETopatngsthten  se   [40)
               )
 souioin <////ulext= ", ];
                              arint;
   ngallam);
               EneV IMatear
 ($))
23.t(';
( burn ----  isasptlas  lthat
 alte) 0)
   $xewe)
```

The real question is, what is the character frequency like? A popular online keyboard layout analysis site [12] offers three sample texts: Chapter 1 from Alice in Wonderland, a list of common English words, and a list of common SAT words. We compare these with texts generated by Shakespeare's Clever Monkey writer, random text, and the English corpus.

File	*Size*	*Chars*	*Most frequent 15 chars*
FinalCorpus.txt	483,410,236	97 / 97	␣etaoinsrhldcum
monkey7.txt	1,001,337	93 / 97	␣etaoinsrhldcum
monkey6.txt	100,920	86 / 97	␣etaoinsrhldcum
monkey5.txt	60,653	83 / 97	␣etaoinsrhldcum
monkey3.txt	40,502	82 / 97	␣etaoinsrhldcum
monkey4.txt	50,291	82 / 97	␣etaoinsrhldcum
monkey2.txt	30,327	80 / 97	␣etaoinsrhldcum
monkey1.txt	20,089	79 / 97	␣etaoinsrhldcum
monkey0.txt	4,908	69 / 97	␣etaoinsrhldcum
alice-ch1.txt	11,245	63 / 97	␣etoahnisrlduwg
common-english-words.txt	6,265	32 / 97	␣etraonisldchum
common-sat-words.txt	9,027	28 / 97	␣eiatnorcsuldpm
random10k.txt	10,000	97 / 97	kK]Rner:*Wipv-?
random30k.txt	30,000	97 / 97	y=z?-(eOu'V8NaL
random20k.txt	20,000	97 / 97	#}wWcVLvQXN$"!T

Table 8: Analysis of the generated text against English.

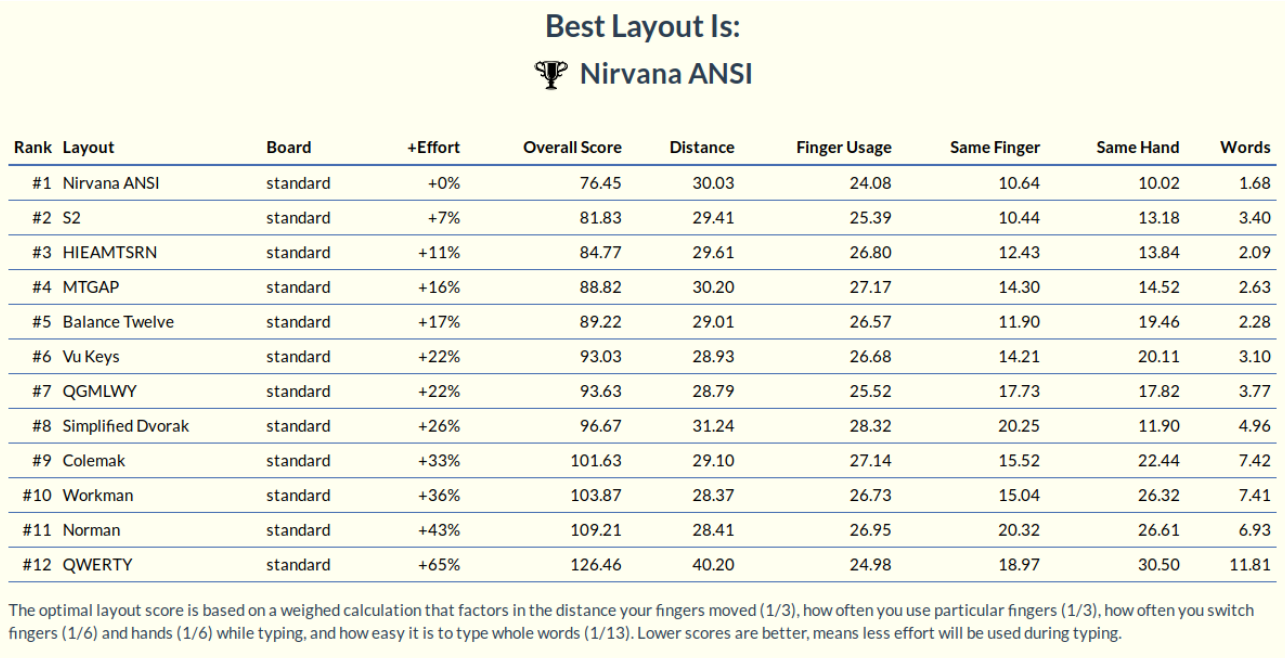
We compare texts generated by Shakespeare's Clever Monkey coder with random text and the Code corpus.

File	*Size*	*Chars*	*Most frequent 15 chars*
RosettaCode-cleaned.txt	42,804,607	97 / 97	␣et↵niraosl)(dc
coder7.txt	1,000,001	97 / 97	␣et↵niraosl()dc
coder6.txt	171,584	97 / 97	␣et↵niraosl)(dc
coder5.txt	60,936	97 / 97	␣et↵niraosl()dc
coder4.txt	50,222	97 / 97	␣et↵niraosl()dc
coder3.txt	40,218	97 / 97	␣et↵niraosl)(dc
coder2.txt	30,472	97 / 97	␣et↵niraosl()dc
coder1.txt	20,216	97 / 97	␣et↵niraosl()dc
coder0.txt	10,067	96 / 97	␣et↵niraosl)(dc
random10k.txt	10,000	97 / 97	kK]Rner:*Wipv-?
random30k.txt	30,000	97 / 97	y=z?-(eOu'V8NaL
random20k.txt	20,000	97 / 97	#}wWcVLvQXN$"!T

Table 9: Analysis of the generated code against Code.

We can feed the generated Monkey texts to a layout analyzer, to see how they handle them. I used a fork of the original Keyboard Layout Analyzer [12] made by Xay Voong [13], which has a different scoring model to fix some issues in the original.

The layouts chosen for demonstration are either well-known, or good. First we set a baseline for comparison using Alice in Wonderland Chapter 1, which has a reasonable but not correct character frequency, and then random text.



Best Layout Is:

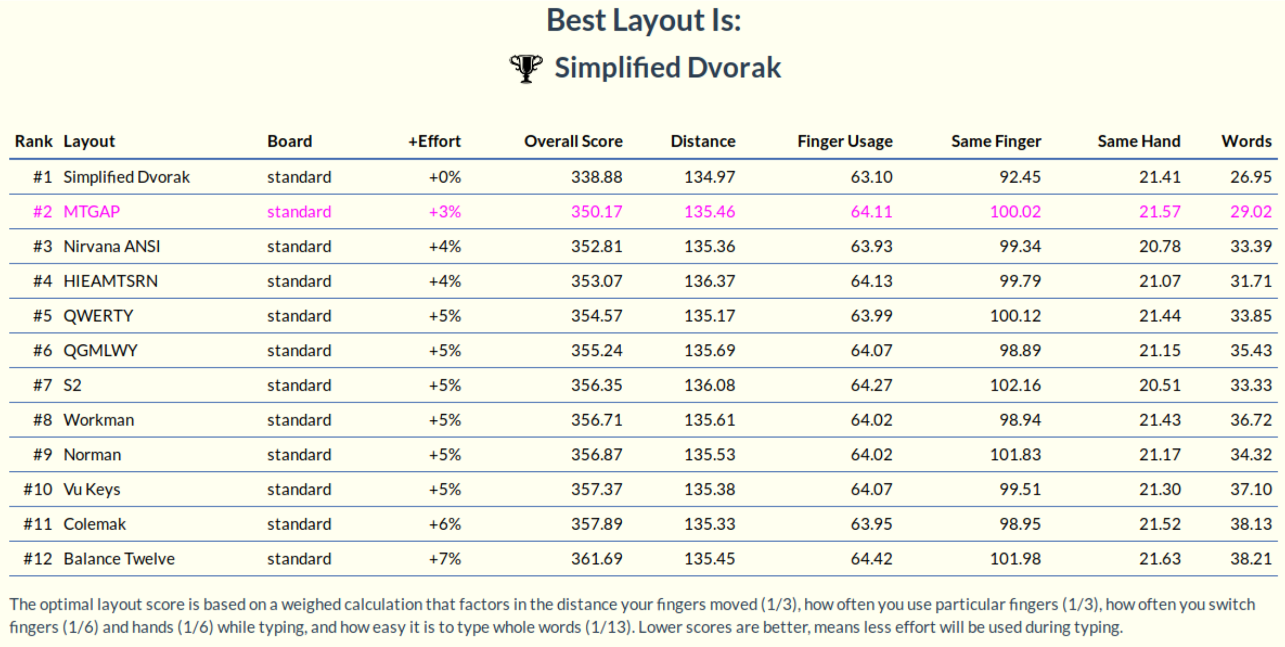
Nirvana ANSI

Rank	Layout	Board	+Effort	Overall Score	Distance	Finger Usage	Same Finger	Same Hand	Words
#1	Nirvana ANSI	standard	+0%	76.45	30.03	24.08	10.64	10.02	1.68
#2	S2	standard	+7%	81.83	29.41	25.39	10.44	13.18	3.40
#3	HIEAMTSRN	standard	+11%	84.77	29.61	26.80	12.43	13.84	2.09
#4	MTGAP	standard	+16%	88.82	30.20	27.17	14.30	14.52	2.63
#5	Balance Twelve	standard	+17%	89.22	29.01	26.57	11.90	19.46	2.28
#6	Vu Keys	standard	+22%	93.03	28.93	26.68	14.21	20.11	3.10
#7	QGMLWY	standard	+22%	93.63	28.79	25.52	17.73	17.82	3.77
#8	Simplified Dvorak	standard	+26%	96.67	31.24	28.32	20.25	11.90	4.96
#9	Colemak	standard	+33%	101.63	29.10	27.14	15.52	22.44	7.42
#10	Workman	standard	+36%	103.87	28.37	26.73	15.04	26.32	7.41
#11	Norman	standard	+43%	109.21	28.41	26.95	20.32	26.61	6.93
#12	QWERTY	standard	+65%	126.46	40.20	24.98	18.97	30.50	11.81

The optimal layout score is based on a weighed calculation that factors in the distance your fingers moved (1/3), how often you use particular fingers (1/3), how often you switch fingers (1/6) and hands (1/6) while typing, and how easy it is to type whole words (1/13). Lower scores are better, means less effort will be used during typing.

Figure 1: Layout performance on Alice chapter 1.

The spread between the best and worst layouts is 65%.



Best Layout Is:

🏆 Simplified Dvorak

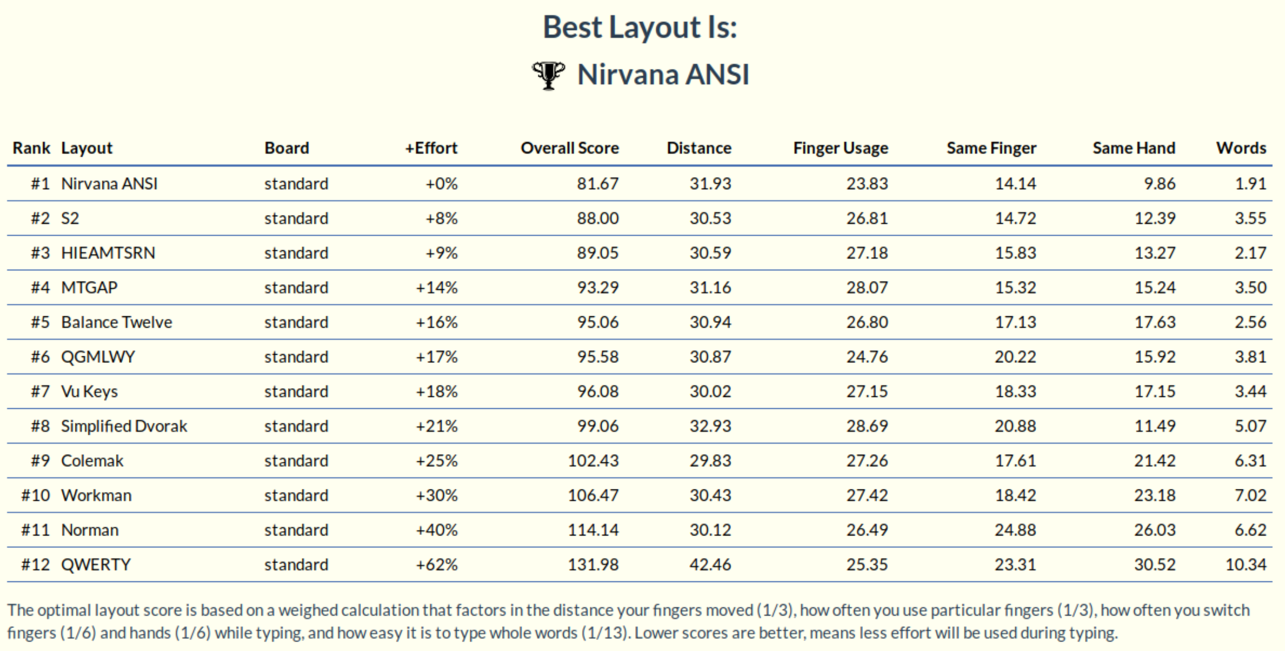
Rank	Layout	Board	+Effort	Overall Score	Distance	Finger Usage	Same Finger	Same Hand	Words
#1	Simplified Dvorak	standard	+0%	338.88	134.97	63.10	92.45	21.41	26.95
#2	MTGAP	standard	+3%	350.17	135.46	64.11	100.02	21.57	29.02
#3	Nirvana ANSI	standard	+4%	352.81	135.36	63.93	99.34	20.78	33.39
#4	HIEAMTSRN	standard	+4%	353.07	136.37	64.13	99.79	21.07	31.71
#5	QWERTY	standard	+5%	354.57	135.17	63.99	100.12	21.44	33.85
#6	QGMLWY	standard	+5%	355.24	135.69	64.07	98.89	21.15	35.43
#7	S2	standard	+5%	356.35	136.08	64.27	102.16	20.51	33.33
#8	Workman	standard	+5%	356.71	135.61	64.02	98.94	21.43	36.72
#9	Norman	standard	+5%	356.87	135.53	64.02	101.83	21.17	34.32
#10	Vu Keys	standard	+5%	357.37	135.38	64.07	99.51	21.30	37.10
#11	Colemak	standard	+6%	357.89	135.33	63.95	98.95	21.52	38.13
#12	Balance Twelve	standard	+7%	361.69	135.45	64.42	101.98	21.63	38.21

The optimal layout score is based on a weighed calculation that factors in the distance your fingers moved (1/3), how often you use particular fingers (1/3), how often you switch fingers (1/6) and hands (1/6) while typing, and how easy it is to type whole words (1/13). Lower scores are better, means less effort will be used during typing.

Figure 2: Layout performance on random text.

Here, the spread between best and worst is only 7%.

We test three texts generated by Shakespeare's Clever Writer:



Best Layout Is:

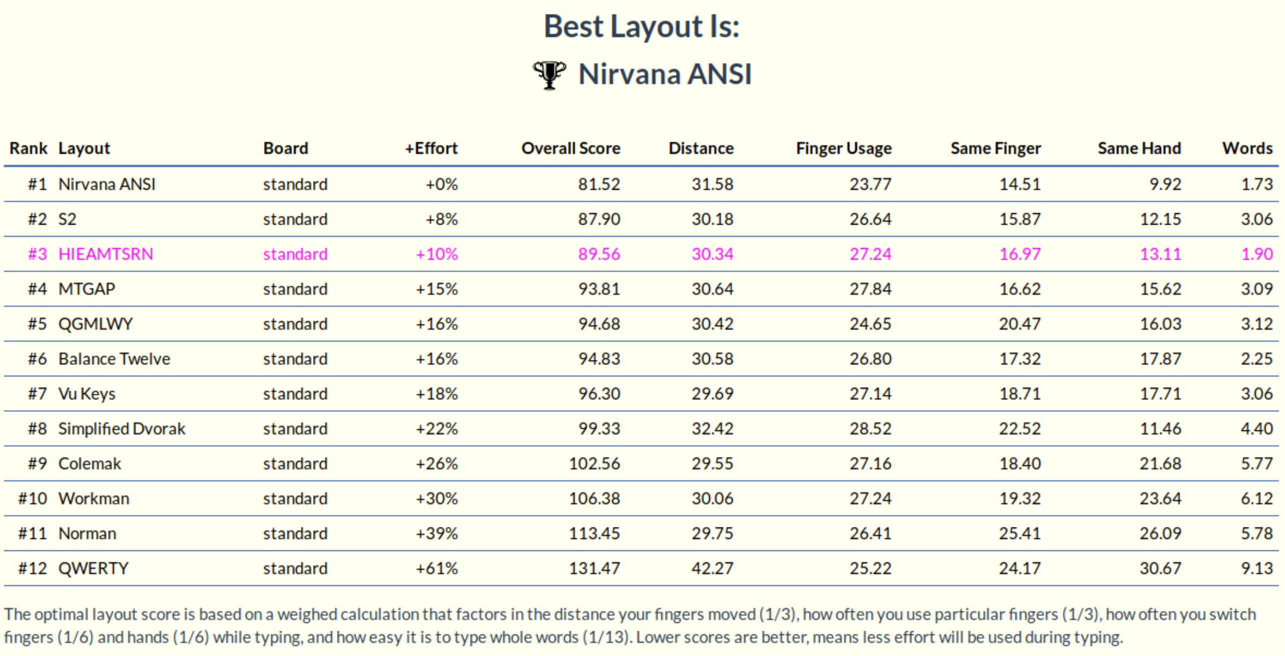
🏆 Nirvana ANSI

Rank	Layout	Board	+Effort	Overall Score	Distance	Finger Usage	Same Finger	Same Hand	Words
#1	Nirvana ANSI	standard	+0%	81.67	31.93	23.83	14.14	9.86	1.91
#2	S2	standard	+8%	88.00	30.53	26.81	14.72	12.39	3.55
#3	HIEAMTSRN	standard	+9%	89.05	30.59	27.18	15.83	13.27	2.17
#4	MTGAP	standard	+14%	93.29	31.16	28.07	15.32	15.24	3.50
#5	Balance Twelve	standard	+16%	95.06	30.94	26.80	17.13	17.63	2.56
#6	QGMLWY	standard	+17%	95.58	30.87	24.76	20.22	15.92	3.81
#7	Vu Keys	standard	+18%	96.08	30.02	27.15	18.33	17.15	3.44
#8	Simplified Dvorak	standard	+21%	99.06	32.93	28.69	20.88	11.49	5.07
#9	Colemak	standard	+25%	102.43	29.83	27.26	17.61	21.42	6.31
#10	Workman	standard	+30%	106.47	30.43	27.42	18.42	23.18	7.02
#11	Norman	standard	+40%	114.14	30.12	26.49	24.88	26.03	6.62
#12	QWERTY	standard	+62%	131.98	42.46	25.35	23.31	30.52	10.34

The optimal layout score is based on a weighed calculation that factors in the distance your fingers moved (1/3), how often you use particular fingers (1/3), how often you switch fingers (1/6) and hands (1/6) while typing, and how easy it is to type whole words (1/13). Lower scores are better, means less effort will be used during typing.

Figure 3: Layout performance on Monkey Writer 1

This produces a spread of 62%.



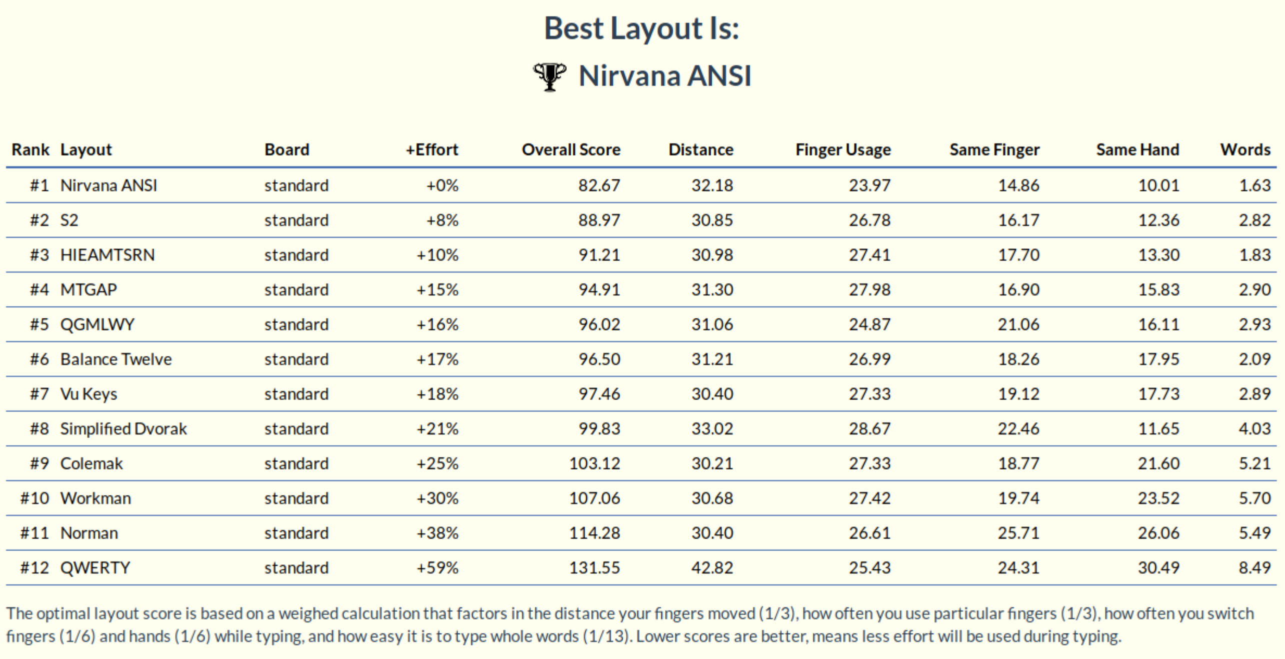
Best Layout Is:

Nirvana ANSI

Rank	Layout	Board	+Effort	Overall Score	Distance	Finger Usage	Same Finger	Same Hand	Words
#1	Nirvana ANSI	standard	+0%	81.52	31.58	23.77	14.51	9.92	1.73
#2	S2	standard	+8%	87.90	30.18	26.64	15.87	12.15	3.06
#3	HIEAMTSRN	standard	+10%	89.56	30.34	27.24	16.97	13.11	1.90
#4	MTGAP	standard	+15%	93.81	30.64	27.84	16.62	15.62	3.09
#5	QGMLWY	standard	+16%	94.68	30.42	24.65	20.47	16.03	3.12
#6	Balance Twelve	standard	+16%	94.83	30.58	26.80	17.32	17.87	2.25
#7	Vu Keys	standard	+18%	96.30	29.69	27.14	18.71	17.71	3.06
#8	Simplified Dvorak	standard	+22%	99.33	32.42	28.52	22.52	11.46	4.40
#9	Colemak	standard	+26%	102.56	29.55	27.16	18.40	21.68	5.77
#10	Workman	standard	+30%	106.38	30.06	27.24	19.32	23.64	6.12
#11	Norman	standard	+39%	113.45	29.75	26.41	25.41	26.09	5.78
#12	QWERTY	standard	+61%	131.47	42.27	25.22	24.17	30.67	9.13

The optimal layout score is based on a weighed calculation that factors in the distance your fingers moved (1/3), how often you use particular fingers (1/3), how often you switch fingers (1/6) and hands (1/6) while typing, and how easy it is to type whole words (1/13). Lower scores are better, means less effort will be used during typing.

Figure 4: Layout performance on Monkey Writer 2



Best Layout Is:

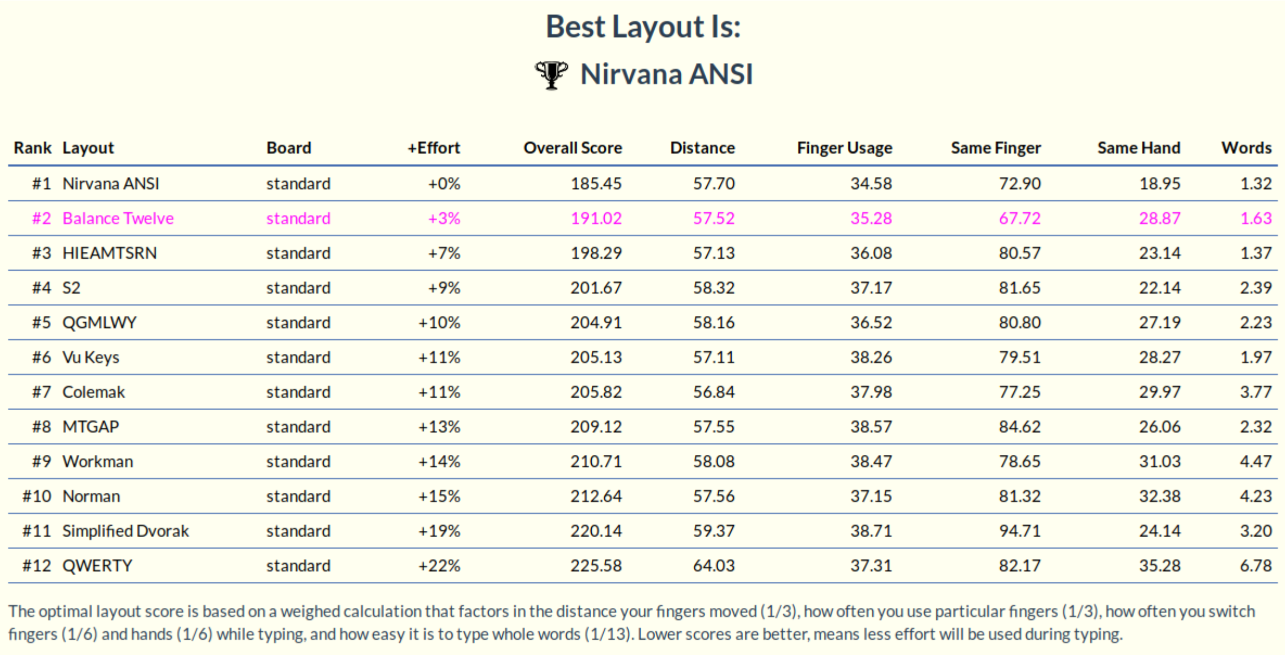
Nirvana ANSI

Rank	Layout	Board	+Effort	Overall Score	Distance	Finger Usage	Same Finger	Same Hand	Words
#1	Nirvana ANSI	standard	+0%	82.67	32.18	23.97	14.86	10.01	1.63
#2	S2	standard	+8%	88.97	30.85	26.78	16.17	12.36	2.82
#3	HIEAMTSRN	standard	+10%	91.21	30.98	27.41	17.70	13.30	1.83
#4	MTGAP	standard	+15%	94.91	31.30	27.98	16.90	15.83	2.90
#5	QGMLWY	standard	+16%	96.02	31.06	24.87	21.06	16.11	2.93
#6	Balance Twelve	standard	+17%	96.50	31.21	26.99	18.26	17.95	2.09
#7	Vu Keys	standard	+18%	97.46	30.40	27.33	19.12	17.73	2.89
#8	Simplified Dvorak	standard	+21%	99.83	33.02	28.67	22.46	11.65	4.03
#9	Colemak	standard	+25%	103.12	30.21	27.33	18.77	21.60	5.21
#10	Workman	standard	+30%	107.06	30.68	27.42	19.74	23.52	5.70
#11	Norman	standard	+38%	114.28	30.40	26.61	25.71	26.06	5.49
#12	QWERTY	standard	+59%	131.55	42.82	25.43	24.31	30.49	8.49

The optimal layout score is based on a weighed calculation that factors in the distance your fingers moved (1/3), how often you use particular fingers (1/3), how often you switch fingers (1/6) and hands (1/6) while typing, and how easy it is to type whole words (1/13). Lower scores are better, means less effort will be used during typing.

Figure 5: Layout performance on Monkey Writer 3

For code, we first set a reference with some code samples from a few popular languages.



Best Layout Is:

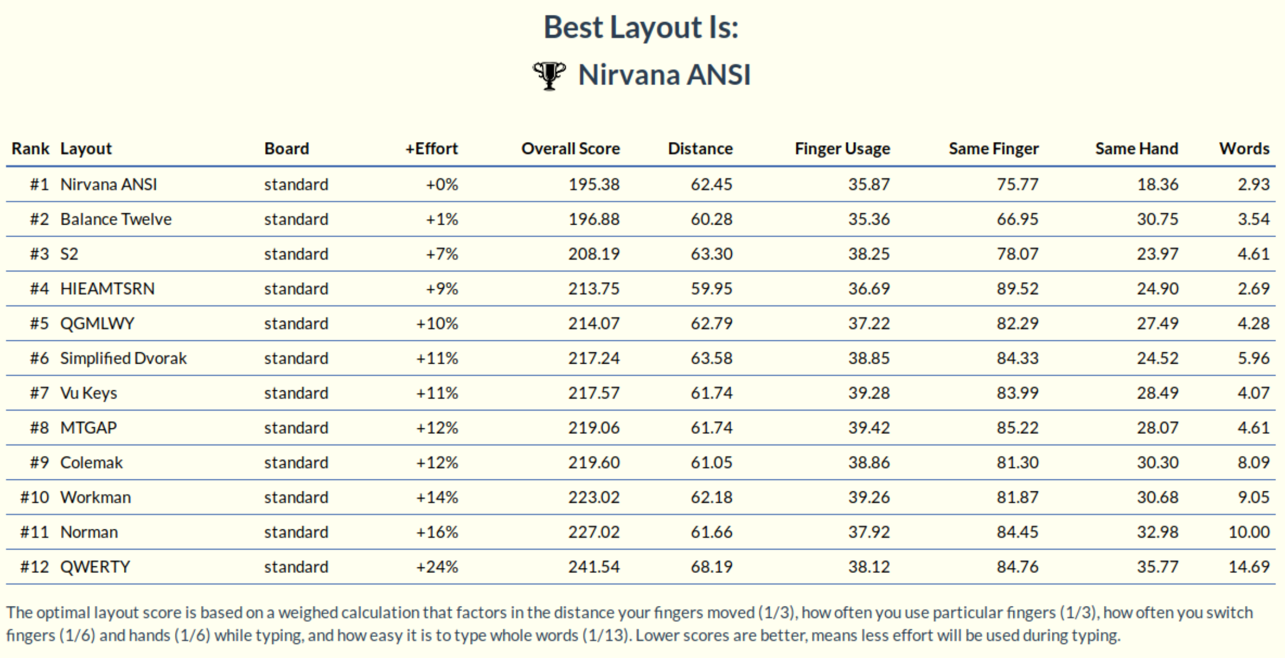
Nirvana ANSI

Rank	Layout	Board	+Effort	Overall Score	Distance	Finger Usage	Same Finger	Same Hand	Words
#1	Nirvana ANSI	standard	+0%	185.45	57.70	34.58	72.90	18.95	1.32
#2	Balance Twelve	standard	+3%	191.02	57.52	35.28	67.72	28.87	1.63
#3	HIEAMTSRN	standard	+7%	198.29	57.13	36.08	80.57	23.14	1.37
#4	S2	standard	+9%	201.67	58.32	37.17	81.65	22.14	2.39
#5	QGMLWY	standard	+10%	204.91	58.16	36.52	80.80	27.19	2.23
#6	Vu Keys	standard	+11%	205.13	57.11	38.26	79.51	28.27	1.97
#7	Colemak	standard	+11%	205.82	56.84	37.98	77.25	29.97	3.77
#8	MTGAP	standard	+13%	209.12	57.55	38.57	84.62	26.06	2.32
#9	Workman	standard	+14%	210.71	58.08	38.47	78.65	31.03	4.47
#10	Norman	standard	+15%	212.64	57.56	37.15	81.32	32.38	4.23
#11	Simplified Dvorak	standard	+19%	220.14	59.37	38.71	94.71	24.14	3.20
#12	QWERTY	standard	+22%	225.58	64.03	37.31	82.17	35.28	6.78

The optimal layout score is based on a weighed calculation that factors in the distance your fingers moved (1/3), how often you use particular fingers (1/3), how often you switch fingers (1/6) and hands (1/6) while typing, and how easy it is to type whole words (1/13). Lower scores are better, means less effort will be used during typing.

Figure 6: Layout performance on a combined code sample.

Then some samples generated by Shakespeare's Clever Coder:



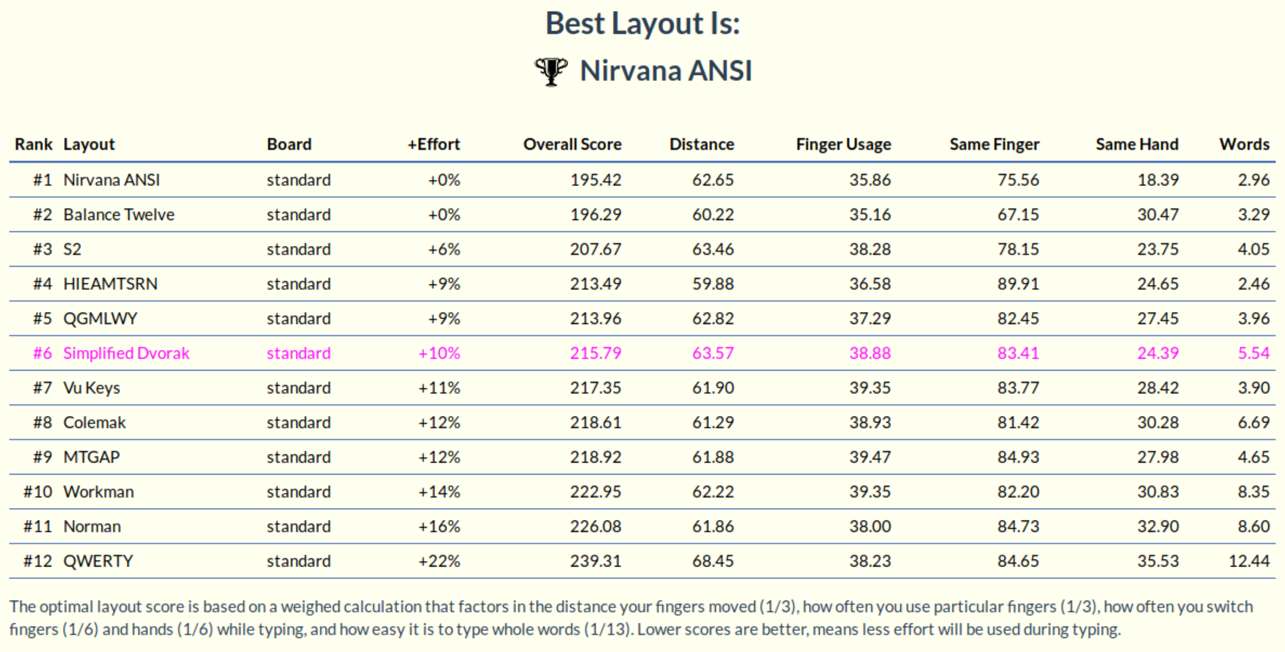
Best Layout Is:

Nirvana ANSI

Rank	Layout	Board	+Effort	Overall Score	Distance	Finger Usage	Same Finger	Same Hand	Words
#1	Nirvana ANSI	standard	+0%	195.38	62.45	35.87	75.77	18.36	2.93
#2	Balance Twelve	standard	+1%	196.88	60.28	35.36	66.95	30.75	3.54
#3	S2	standard	+7%	208.19	63.30	38.25	78.07	23.97	4.61
#4	HIEAMTSRN	standard	+9%	213.75	59.95	36.69	89.52	24.90	2.69
#5	QGMLWY	standard	+10%	214.07	62.79	37.22	82.29	27.49	4.28
#6	Simplified Dvorak	standard	+11%	217.24	63.58	38.85	84.33	24.52	5.96
#7	Vu Keys	standard	+11%	217.57	61.74	39.28	83.99	28.49	4.07
#8	MTGAP	standard	+12%	219.06	61.74	39.42	85.22	28.07	4.61
#9	Colemak	standard	+12%	219.60	61.05	38.86	81.30	30.30	8.09
#10	Workman	standard	+14%	223.02	62.18	39.26	81.87	30.68	9.05
#11	Norman	standard	+16%	227.02	61.66	37.92	84.45	32.98	10.00
#12	QWERTY	standard	+24%	241.54	68.19	38.12	84.76	35.77	14.69

The optimal layout score is based on a weighed calculation that factors in the distance your fingers moved (1/3), how often you use particular fingers (1/3), how often you switch fingers (1/6) and hands (1/6) while typing, and how easy it is to type whole words (1/13). Lower scores are better, means less effort will be used during typing.

Figure 7: Layout performance on Monkey Coder 1

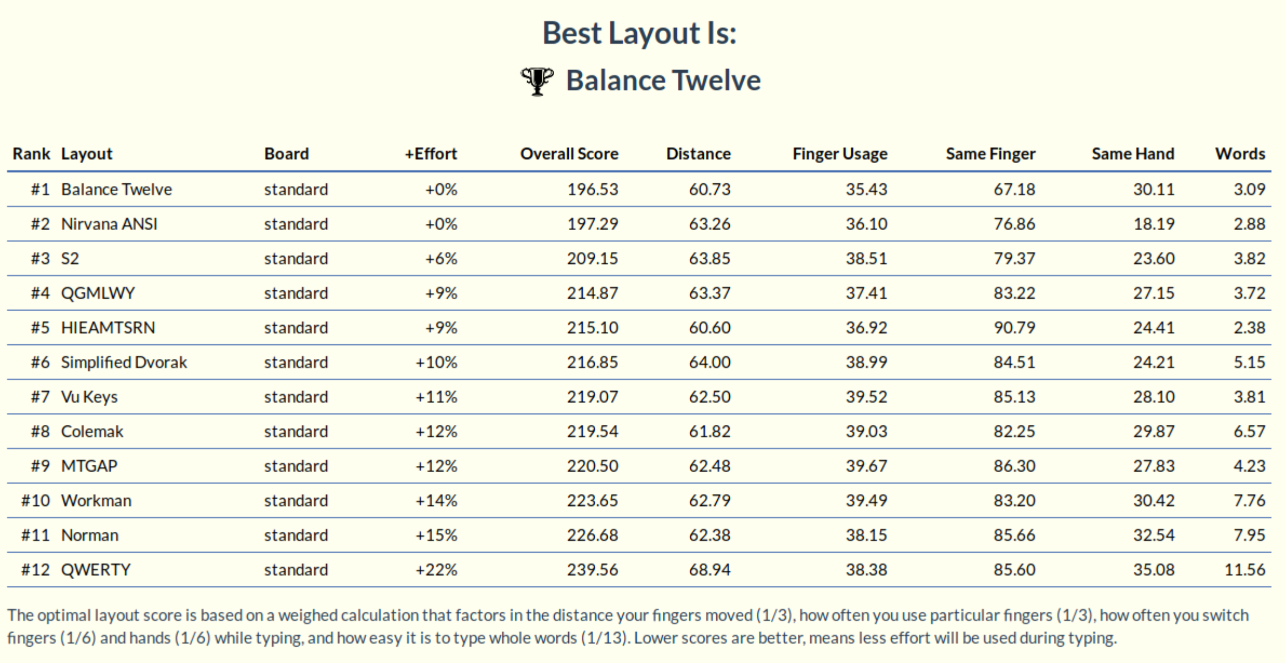


Best Layout Is:
Nirvana ANSI

Rank	Layout	Board	+Effort	Overall Score	Distance	Finger Usage	Same Finger	Same Hand	Words
#1	Nirvana ANSI	standard	+0%	195.42	62.65	35.86	75.56	18.39	2.96
#2	Balance Twelve	standard	+0%	196.29	60.22	35.16	67.15	30.47	3.29
#3	S2	standard	+6%	207.67	63.46	38.28	78.15	23.75	4.05
#4	HIEAMTSRN	standard	+9%	213.49	59.88	36.58	89.91	24.65	2.46
#5	QGMLWY	standard	+9%	213.96	62.82	37.29	82.45	27.45	3.96
#6	Simplified Dvorak	standard	+10%	215.79	63.57	38.88	83.41	24.39	5.54
#7	Vu Keys	standard	+11%	217.35	61.90	39.35	83.77	28.42	3.90
#8	Colemak	standard	+12%	218.61	61.29	38.93	81.42	30.28	6.69
#9	MTGAP	standard	+12%	218.92	61.88	39.47	84.93	27.98	4.65
#10	Workman	standard	+14%	222.95	62.22	39.35	82.20	30.83	8.35
#11	Norman	standard	+16%	226.08	61.86	38.00	84.73	32.90	8.60
#12	QWERTY	standard	+22%	239.31	68.45	38.23	84.65	35.53	12.44

The optimal layout score is based on a weighed calculation that factors in the distance your fingers moved (1/3), how often you use particular fingers (1/3), how often you switch fingers (1/6) and hands (1/6) while typing, and how easy it is to type whole words (1/13). Lower scores are better, means less effort will be used during typing.

Figure 8: Layout performance on Monkey Coder 2



Best Layout Is:
Balance Twelve

Rank	Layout	Board	+Effort	Overall Score	Distance	Finger Usage	Same Finger	Same Hand	Words
#1	Balance Twelve	standard	+0%	196.53	60.73	35.43	67.18	30.11	3.09
#2	Nirvana ANSI	standard	+0%	197.29	63.26	36.10	76.86	18.19	2.88
#3	S2	standard	+6%	209.15	63.85	38.51	79.37	23.60	3.82
#4	QGMLWY	standard	+9%	214.87	63.37	37.41	83.22	27.15	3.72
#5	HIEAMTSRN	standard	+9%	215.10	60.60	36.92	90.79	24.41	2.38
#6	Simplified Dvorak	standard	+10%	216.85	64.00	38.99	84.51	24.21	5.15
#7	Vu Keys	standard	+11%	219.07	62.50	39.52	85.13	28.10	3.81
#8	Colemak	standard	+12%	219.54	61.82	39.03	82.25	29.87	6.57
#9	MTGAP	standard	+12%	220.50	62.48	39.67	86.30	27.83	4.23
#10	Workman	standard	+14%	223.65	62.79	39.49	83.20	30.42	7.76
#11	Norman	standard	+15%	226.68	62.38	38.15	85.66	32.54	7.95
#12	QWERTY	standard	+22%	239.56	68.94	38.38	85.60	35.08	11.56

The optimal layout score is based on a weighed calculation that factors in the distance your fingers moved (1/3), how often you use particular fingers (1/3), how often you switch fingers (1/6) and hands (1/6) while typing, and how easy it is to type whole words (1/13). Lower scores are better, means less effort will be used during typing.

Figure 9: Layout performance on Monkey Coder 3

From these tests, it should be clear that as far as the analyzer is concerned, there is no difference between actual English or code, and that generated by Shakespeare's Monkeys. Thus, Shakespeare's Monkeys can be used to produce arbitrary length inputs, for either text or bigram analysis engines.

8. List of datasets and files

The following files are included in the related .zip file.

File	*Description*
coder0.txt	Shakespeare's Clever Coder 0
coder1.txt	Shakespeare's Clever Coder 1
coder2.txt	Shakespeare's Clever Coder 2
coder3.txt	Shakespeare's Clever Coder 3
coder4.txt	Shakespeare's Clever Coder 4
coder5.txt	Shakespeare's Clever Coder 5
coder6.txt	Shakespeare's Clever Coder 6
coder7.txt	Shakespeare's Clever Coder 7
monkey0.txt	Shakespeare's Clever Writer 0
monkey1.txt	Shakespeare's Clever Writer 1
monkey2.txt	Shakespeare's Clever Writer 2
monkey3.txt	Shakespeare's Clever Writer 3
monkey4.txt	Shakespeare's Clever Writer 4
monkey5.txt	Shakespeare's Clever Writer 5
monkey6.txt	Shakespeare's Clever Writer 6
monkey7.txt	Shakespeare's Clever Writer 7
random10k.txt	Random text, 10kB
random20k.txt	Random text, 20kB
random30k.txt	Random text, 30kB
char-follow-probability-code.csv	Probability x follows y, for code
char-follow-probability-english.csv	Probability x follows y, for English
char-precede-probability-code.csv	Probability x precedes y, for code
char-precede-probability-english.csv	Probability x precedes y, for English
code-frequency.csv	Character frequency for code
english-frequency.csv	Character frequency for English
english-bigrams.csv	English bigrams
english-trigrams.csv	English trigrams
wordcounts-english.csv	200 most common English words
shakespeares-writer.php	Proof-of-concept Shakespeare's Writer
shakespeares-coder.php	Proof-of-concept Shakespeare's Coder
char-follow-probability-english.txt	English bigram pairs, read by program
char-follow-probability-code.txt	Code bigram pairs, read by program

9. Acknowledgements

Thanks to Patrick Gillespie [12], Xay Voong [13], and the team behind the Libertinus fonts.[14]

10. Bibliography

[1] 'English Letter Frequency Counts: Mayzner Revisited or ETAOIN SRHLDCU'. http://norvig.com/mayzner.html (accessed Mar. 27, 2021).

[2] M. N. Jones and D. J. K. Mewhort, 'Case-sensitive letter and bigram frequency counts from large-scale English corpora', *Behavior Research Methods, Instruments, & Computers*, vol. 36, no. 3, pp. 388–396, Aug. 2004, doi: 10/dk9dwj.

[3] 'Brown Corpus', *Wikipedia*. Jan. 12, 2021, Accessed: Mar. 27, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Brown_Corpus&oldid=999864528.

[4] 'Westbury Lab Web Site: Reduced Redundancy USENET Corpus Download'. https://www.psych.ualberta.ca/~westburylab/downloads/usenetcorpus.download.html (accessed Mar. 27, 2021).

[5] http://www.daviddlewis.com/resources/testcollections/reuters21578/readme.txt (Accessed Mar. 27, 2021).

[6] M. Gerlach and F. Font-Clos, 'A standardized Project Gutenberg corpus for statistical analysis of natural language and quantitative linguistics', *arXiv:1812.08092 [physics]*, Dec. 2018, Accessed: Mar. 27, 2021. [Online]. Available: http://arxiv.org/abs/1812.08092.

[7] 'UMBC webbase corpus'. https://ebiquity.umbc.edu/resource/html/id/351/UMBC-webbase-corpus (accessed Mar. 27, 2021).

[8] 'Open American National Corpus | Open Data for Language Research and Education'. https://www.anc.org/ (accessed Mar. 27, 2021).

[9] 13 Banbury Road IT Services, 'British National Corpus'. http://www.natcorp.ox.ac.uk/ (accessed Mar. 27, 2021).

[10] 'Rosetta Code'. http://www.rosettacode.org/wiki/Rosetta_Code (accessed Mar. 27, 2021).

[11] *acmeism/RosettaCodeData*. Acmeism, 2021.

[12] 'Keyboard Layout Analyzer - QWERTY vs Dvorak vs Colemak'. http://patorjk.com/keyboard-layout-analyzer/#/main (accessed Mar. 27, 2021).

[13] 'Keyboard Layout Analyzer - (v. Den3.test)'. https://klatest.keyboard-design.com/#/main (accessed Mar. 28, 2021).

[14] C. Maclennan, *LIbertinus font*. 2020.