Project 1

Kelvin Rosario

September 10, 2025

1. Background

We've been tasked with reversing a relatively short piece of text that was encrypted using a substitution cipher. We're going to try to break it via a letter frequency attack, let's see if we're successful!

> lrvmnir bpr sumvbwvr jx bpr lmiwv yjeryrkbi jx qmbm wi bpr xjvni mkd ymibrut jx irhx wi bpr riirkvr jx ymbinlmtmipw utn qmumbr dj w ipmhh but bj rhnvwdmbr bpr yjeryrkbi jx bpr qmbm mvvjudwko bj yt wkbrusurbmbwjk lmird jk xjubt trmui jx ibndt
>
> wb wi kjb mk rmit bmiq bj rashmwk rmvp yjeryrkb mkd wbi iwokwxwvmkvr mkd ijyr ynib urymwk nkrashmwkrd bj ower m vjyshrbr rashmkmbwjk jkr cjnhd pmer bj lr fnmhwxwrd mkd wkiswurd bj invp mk rabrkb bpmb pr vjnhd urmvp bpr ibmbr jx rkhwopbrkrd ywkd vmsmlhr jx urvjokwgwko ijnkdhrii ijnkd mkd ipmsrhrii ipmsr w dj kjb drry ytirhx bpr xwkmh mnbpjuwbt lnb yt rasruwrkvr cwbp qmbm pmi hrxb kj djnlb bpmb bpr xjhhjcwko wi bpr sujsru msshwvmbwjk mkd wkbrusurbmbwjk w jxxru yt bprjuwri wk bpr pjsr bpmb bpr riirkvr jx jqwkmcmk qmumbr cwhh urymwk wkbmvb

2. Introduction

At this stage, we didn't immediately see the need to get started on writing a program just yet, so we sorted the letter frequencies of both the cipher text and of English in Google Sheets to allow us to immediately begin decryption attempts.[1]

[1] We do eventually get around to writing a small program once the functionality we require becomes clear.

R	84	0.1300
B	68	0.1053
M	62	0.0960
K	49	0.0759
J	48	0.0743
W	47	0.0728

Figure 1: Cipher text frequencies

It may not seem like much, but this critical step allowed us to build some confidence on an otherwise daunting task. Could we really do this? We immediately opened up Visual Studio Code and tried a few substitutions with just `CTRL + F`, and replace all.

E	0.127
T	0.0906
A	0.0817
O	0.0751
I	0.0697
N	0.0675

Figure 2: English letter frequencies

3. Early Decryption Attempts

3.1 r –> e

We see that both cipher text letter `r` and english letter `e` seem to have matching frequencies, so let's try those first.

Table 1: Replacements so far

Cipher Letter	English Letter
r	e

levmnie bpe sumvbwve jx bpe lmiwv yjeeyekbi jx qmbm wi bpe xjvni mkd ymibeut jx iehx wi bpe eiiekve jx ymbinlmtmipw utn qmumbe dj w ipmhh but bj ehnvwdmbe bpe yjeeyekbi jx bpe qmbm mvvjudwko bj yt wkbeusuebmbwjk lmied jk xjubt temui jx ibndt

wb wi kjb mk emit bmiq bj eashmwk emvp yjeeyekb mkd wbi iwokwxwvmkve mkd ijye ynib ueymwk nkeashmwked bj owee m vjyshebe eashmkmbwjk jke cjnhd pmee bj le fnmhwxwed mkd wkiswued bj invp mk eabekb bpmb pe vjnhd uemvp bpe ibmbe jx ekhwopbeked ywkd vmsmlhe jx uevjokwgwko ijnkdheii ijnkd mkd ipmseheii ipmse w dj kjb deey ytiehx bpe xwkmh mnbpjuwbt lnb yt easeuwekve cwbp qmbm pmi hexb kj djnlb bpmb bpe xjhhjcwko wi bpe sujseu msshwvmbwjk mkd wkbeusuebmbwjk w jxxeu yt bpejuwei wk bpe pjse bpmb bpe eiiekve jx jqwkmcmk qmumbe cwhh ueymwk wkbmvb

Nothing that we can really make out as English yet. We do see the word 'but' appear however it's too early for it to actually be a decrypted word. Let's try a few more substitutions that seem to match up.

3.2 b –> t

Table 2: Replacements so far

Cipher Letter	English Letter
r	e
b	t

levmnie tpe sumvtwve jx tpe lmiwv yjeeyekti jx qmtm wi tpe xjvni mkd ymiteut jx iehx wi tpe eiiekve jx ymtinlmtmipw utn qmumte dj w ipmhh tut tj ehnvwdmte tpe yjeeyekti jx tpe qmtm mvvjudwko tj yt wkteusuetmtwjk lmied jk xjutt temui jx itndt

wt wi kjt mk emit tmiq tj eashmwk emvp yjeeyekt mkd wti iwokwxwvmkve mkd ijye ynit ueymwk nkeashmwked tj owee m vjyshete eashmkmtwjk jke cjnhd pmee tj le fnmhwxwed mkd wkiswued tj invp mk eatekt tpmt pe vjnhd uemvp tpe itmte jx ekhwopteked ywkd vmsmlhe jx uevjokwgwko ijnkdheii ijnkd mkd ipmseheii ipmse w dj kjt deey ytiehx tpe xwkmh mntpjuwtt lnt yt easeuwekve cwtp qmtm pmi hext kj djnlt tpmt tpe xjhhjcwko wi tpe sujseu msshwvmtwjk mkd wkteusuetmtwjk w jxxeu yt tpejuwei wk tpe pjse tpmt tpe eiiekve jx jqwkmcmk qmumte cwhh ueymwk wktmvt

3.3 m −> a

Table 3: Replacements so far

Cipher Letter	English Letter
r	e
b	t
m	a

levanie tpe suavtwve jx tpe laiwv yjeeyekti jx qata wi tpe xjvni akd yaiteut jx iehx wi tpe eiiekve jx yatinlataipw utn qauate dj w ipahh tut tj ehnvwdate tpe yjeeyekti jx tpe qata avvjudwko tj yt wkteusuetatwjk laied jk xjutt teaui jx itndt

wt wi kjt ak eait taiq tj eashawk eavp yjeeyekt akd wti iwokwxwvakve akd ijye ynit ueyawk nkeashawked tj owee a vjyshete eashakatwjk jke cjnhd paee tj le fnahwxwed akd wkiswued tj invp ak eatekt tpat pe vjnhd ueavp tpe itate jx ekhwopteked ywkd vasalhe jx uevjokwgwko ijnkdheii ijnkd akd ipaseheii ipase w dj kjt deey ytiehx tpe xwkah antpjuwtt lnt yt easeuwekve cwtp qata pai hext kj djnlt tpat tpe xjhhjcwko wi tpe sujseu asshwvatwjk akd wkteusuetatwjk w jxxeu yt tpejuwei wk tpe pjse tpat tpe eiiekve jx jqwkacak qauate cwhh ueyawk wktavt

3.4 k −> o

At this stage we've still got a few substitutions that are likely to match up but we've got a problem: the frequencies of the remaining letters are so close to each other that any one of them could be the correct candidate for substitution with 'o'. We might've also inadvertently stumbled upon an easy way to defeat this attack: make sure the cipher text frequencies are all close to each other.

There's no way to know in advance which the correct substitution might be, so we naively try mapping 'k' to 'o'.

Table 4: Replacements so far

Cipher Letter	English Letter
r	e
b	t
m	a
k	o

levanie tpe suavtwve jx tpe laiwv yjeeyeoti jx qata wi tpe xjvni aod yaiteut jx iehx wi tpe eiieove jx yatinlataipw utn qauate dj w

> ipahh tut tj ehnvwdate tpe yjeeyeoti jx tpe qata avvjudwoo tj yt woteusuetatwjo laied jo xjutt teaui jx itndt
>
> wt wi ojt ao eait taiq tj eashawo eavp yjeeyeot aod wti iwoowxwvaove aod ijye ynit ueyawo noeashawoed tj owee a vjyshete eashaoatwjo joe cjnhd paee tj le fnahwxwed aod woiswued tj invp ao eateot tpat pe vjnhd ueavp tpe itate jx eohwopteoed ywod vasalhe jx uevjoowgwoo ijnodheii ijnod aod ipaseheii ipase w dj ojt deey ytiehx tpe xwoah antpjuwtt lnt yt easeuweove cwtp qata pai hext oj djnlt tpat tpe xjhhjcwoo wi tpe sujseu asshwvatwjo aod woteusuetatwjo w jxxeu yt tpejuwei wo tpe pjse tpat tpe eiieove jx jqwoacao qauate cwhh ueyawo wotavt

This seems to have gotten us less readable text. Could it be 'j' to 'o' instead?

3.5 j -> o

Table 5: Replacements so far

Cipher Letter	English Letter
r	e
b	t
m	a
j	o

> levanie tpe suavtwve ox tpe laiwv yoeeyekti ox qata wi tpe xovni akd yaiteut ox iehx wi tpe eiiekve ox yatinlataipw utn qauate do w ipahh tut to ehnvwdate tpe yoeeyekti ox tpe qata avvoudwko to yt wkteusuetatwok laied ok xoutt teaui ox itndt
>
> wt wi kot ak eait taiq to eashawk eavp yoeeyekt akd wti iwokwxwvakve akd ioye ynit ueyawk nkeashawked to owee a voyshete eashakatwok oke conhd paee to le fnahwxwed akd wkiswued to invp ak eatekt tpat pe vonhd ueavp tpe itate ox ekhwopteked ywkd vasalhe ox uevookwgwko ionkdheii ionkd akd ipaseheii ipase w do kot deey ytiehx tpe xwkah antpouwtt lnt yt easeuwekve cwtp qata pai hext ko donlt tpat tpe xohhocwko wi tpe suoseu asshwvatwok akd wkteusuetatwok w oxxeu yt tpeouwei wk tpe pose tpat tpe eiiekve ox oqwkacak qauate cwhh ueyawk wktavt

We start seeing 'to' pop up, and we're confident it's actually 'to', because we mapped 'b' to 't' earlier. So we're starting to see something that looks like it's leaning towards English! It looks like this was the correct substitution, so we'll keep trying a few more.

3.6 k –> i

Since ‘k’ was undone in the last step, we wonder what it might be. Trying to match it with the English letter closest in relative frequency would have us map ‘k’ –> ‘i’.

Table 6: Replacements so far

Cipher Letter	English Letter
r	e
b	t
m	a
j	o
k	i

> levanie tpe suavtwve ox tpe laiwv yoeeyeiti ox qata wi tpe xovni aid yaiteut ox iehx wi tpe eiieive ox yatinlataipw utn qauate do w ipahh tut to ehnvwdate tpe yoeeyeiti ox tpe qata avvoudwio to yt witeusuetatwoi laied oi xoutt teaui ox itndt
>
> wt wi iot ai eait taiq to eashawi eavp yoeeyeit aid wti iwoiwxwvaive aid ioye ynit ueyawi nieashawied to owee a voyshete eashaiatwoi oie conhd paee to le fnahwxwed aid wiiswued to invp ai eateit tpat pe vonhd ueavp tpe itate ox eihwopteied ywid vasalhe ox uevooiwgwio ionidheii ionid aid ipaseheii ipase w do iot deey ytiehx tpe xwiah antpouwtt lnt yt easeuweive cwtp qata pai hext io donlt tpat tpe xohhocwio wi tpe suoseu asshwvatwoi aid witeusuetatwoi w oxxeu yt tpeouwei wi tpe pose tpat tpe eiieive ox oqwiacai qauate cwhh ueyawi witavt

This doesn’t seem like a good map. We got ‘iot’ as a decrypted word, and we’re confident that the ‘o’ and the ‘t’ are correct mappings. The next closest English letter would be ‘n’, so let’s try that next.

3.6 k –> n

Table 7: Replacements so far

Cipher Letter	English Letter
r	e
b	t
m	a
j	o
k	n

levanie tpe suavtwve ox tpe laiwv yoeeyenti ox qata wi tpe xovni and yaiteut ox iehx wi tpe eiienve ox yatinlataipw utn qauate do w ipahh tut to ehnvwdate tpe yoeeyenti ox tpe qata avvoudwno to yt wnteusuetatwon laied on xoutt teaui ox itndt

wt wi not an eait taiq to eashawn eavp yoeeyent and wti iwonwxwvanve and ioye ynit ueyawn nneashawned to owee a voyshete eashanatwon one conhd paee to le fnahwxwed and wniswued to invp an eatent tpat pe vonhd ueavp tpe itate ox enhwoptened ywnd vasalhe ox uevoonwgwno ionndheii ionnd and ipaseheii ipase w do not deey ytiehx tpe xwnah antpouwtt lnt yt easeuwenve cwtp qata pai hext no donlt tpat tpe xohhocwno wi tpe suoseu asshwvatwon and wnteusuetatwon w oxxeu yt tpeouwei wn tpe pose tpat tpe eiienve ox oqwnacan qauate cwhh ueyawn wntavt

We just saw 'not' appear! This definitely feels like the right direction so far.

3.7 Taking a look at our frequency tables again

It seems like our frequency tables are becoming less reliable. It's still too early to start guessing so they're still our best bet at the moment. Let's revist them for likely substitutions.

W	47	0.0728
I	41	0.0635
P	30	0.0464
U	24	0.0372
D	23	0.0356
H	23	0.0356

Figure 3: Cipher text frequencies continued

I	0.0697
N	0.0675
S	0.0633
H	0.0609
R	0.0599
D	0.0425

Figure 4: English letter frequencies continued

We see that the next most frequent cipher letter is 'w'. Let's try mapping that to the next most frequent English letter, 'i'.

3.8 w –> i

Table 8: Replacements so far

Cipher Letter	English Letter
r	e
b	t
m	a
j	o
k	n
w	i

levanie tpe suavtive ox tpe laiiv yoeeyenti ox qata ii tpe xovni and yaiteut ox iehx ii tpe eiienve ox yatinlataipi utn qauate do i ipahh tut to ehnvidate tpe yoeeyenti ox tpe qata avvoudino to yt inteusuetation laied on xoutt teaui ox itndt

it ii not an eait taiq to eashain eavp yoeeyent and iti iionixivanve and ioye ynit ueyain nneashained to oiee a voyshete eashanation one conhd paee to le fnahixied and inisiued to invp an eatent tpat pe vonhd ueavp tpe itate ox enhioptened yind vasalhe ox uevoonigino ionndheii ionnd and ipaseheii ipase i do not deey ytiehx tpe xinah antpouitt lnt yt easeuienve citp qata pai hext no donlt tpat tpe xohhocino ii tpe suoseu asshivation and inteusuetation i oxxeu yt tpeouiei in tpe pose tpat tpe eiienve ox oqinacan qauate cihh ueyain intavt

We see the word ‘it’ appear, so this certainly seems like a good substitution! We’ll keep it. The next most frequent letter pairing is cipher letter ‘i’ mapped to ‘s’. Let’s give it a shot.

3.8 i –> s (Oh. . . .no)

Table 9: Replacements so far

Cipher Letter	English Letter
r	e
b	t
m	a
j	o
k	n
w	i
i	s

> levanse tpe suavtsve ox tpe lassv yoeeyents ox qata ss tpe xovns and yasteut ox sehx ss tpe essenve ox yatsnlatasps utn qauate do s spahh tut to ehnvsdate tpe yoeeyents ox tpe qata avvoudsno to yt snteusuetatson lased on xoutt teaus ox stndt
>
> st ss not an east tasq to eashasn eavp yoeeyent and sts ssonsxsvanve and soye ynst ueyasn nneashasned to osee a voyshete eashanatson one conhd paee to le fnahsxsed and snsssued to snvp an eatent tpat pe vonhd ueavp tpe state ox enhsoptened ysnd vasalhe ox uevoonsgsno sonndhess sonnd and spasehess spase s do not deey ytsehx tpe xsnah antpoustt lnt yt easeusenve cstp qata pas hext no donlt tpat tpe xohhocsno ss tpe suoseu asshsvatson and snteusuetatson s oxxeu yt tpeouses sn tpe pose tpat tpe essenve ox oqsnacan qauate cshh ueyasn sntavt

Oh no! We just undid our earlier decryption of ‘w’ to ‘i’. Now we see what we need our program to do. We a way to to track letters that have already been decrypted. We’ll start thinking about how to implement that soon, but for now let’s see how far we can get with an easy trick:

1. Undo w -> i
2. Map i -> s
3. Map w -> i

> levanse tpe suavtive ox tpe lasiv yoeeyents ox qata is tpe xovns and yasteut ox sehx is tpe essenve ox yatsnlataspi utn qauate do i spahh tut to ehnvidate tpe yoeeyents ox tpe qata avvoudino to yt inteusuetation lased on xoutt teaus ox stndt
>
> it is not an east tasq to eashain eavp yoeeyent and its sionixivanve and soye ynst ueyain nneashained to oiee a voyshete eashanation one

> conhd paee to le fnahixied and inssiued to snvp an eatent tpat pe vonhd ueavp tpe state ox enhioptened yind vasalhe ox uevoonigino sonndhess sonnd and spasehess spase i do not deey ytsehx tpe xinah antpouitt lnt yt easeuienve citp qata pas hext no donlt tpat tpe xohhocino is tpe suoseu asshivation and inteusuetation i oxxeu yt tpeouies in tpe pose tpat tpe essenve ox oqinacan qauate cihh ueyain intavt

All of the sudden we have a lot of words we can guess. 'tpe' is probably 'the', so we can guess that 'p' maps onto 'h'.

3.9 p -> h

Table 10: Replacements so far

Cipher Letter	English Letter
r	e
b	t
m	a
j	o
k	n
i	s
w	i
p	h

> levanse the suavtive ox the lasiv yoeeyents ox qata is the xovns and yasteut ox sehx is the essenve ox yatsnlatashi utn qauate do i shahh tut to ehnvidate the yoeeyents ox the qata avvoudino to yt inteusuetation lased on xoutt teaus ox stndt

> it is not an east tasq to eashain eavh yoeeyent and its sionixivanve and soye ynst ueyain nneashained to oiee a voyshete eashanation one conhd haee to le fnahixied and inssiued to snvh an eatent that he vonhd ueavh the state ox enhiohtened yind vasalhe ox uevoonigino sonndhess sonnd and shasehess shase i do not deey ytsehx the xinah anthouitt lnt yt easeuienve cith qata has hext no donlt that the xohhocino is the suoseu asshivation and inteusuetation i oxxeu yt theouies in the hose that the essenve ox oqinacan qauate cihh ueyain intavt

This definitely seems right. Anymore substitutions we can glaze from this so far?

- How about 'theouies'? Perhaps we can map 'u' to 'r' to make 'theories'. It's worth a shot.
- The word 'ox' looks enticing. That's probably 'of'. Let's try mapping 'x' to 'f' as well.

- 'it is not an east tasq'? 't' probably maps onto 'y', and 'q' probably maps onto 'k'.

Let's try these to see what we get.

3.10 Oh....no Part 2: u –> r, x –> f, t –> y, q –> k

Table 11: Replacements so far

Cipher Letter	English Letter
r	e
b	t
m	a
j	o
k	n
i	s
w	i
p	h
u	r
x	f
t	y
q	k

Oh no! Not again! Our mapping of 't' to 'y' undid our mapping of 'b' to 't'. This is really becoming cumbersome now:

> levanse yhe sravyive of yhe lasiv yoeeyenys of kaya is yhe fovns and yasyery of sehf is yhe essenve of yaysnlayashi ryn karaye do i shahh yry yo ehnvidaye yhe yoeeyenys of yhe kaya avvordino yo yy inyersreyayion lased on foryy years of syndy
>
> iy is noy an easy yask yo eashain eavh yoeeyeny and iys sionifivanve and soye ynsy reyain nneashained yo oiee a voysheye eashanayion one conhd haee yo le fnahified and inssired yo snvh an eayeny yhay he vonhd reavh yhe syaye of enhiohyened yind vasalhe of revoonigino sonndhess sonnd and shasehess shase i do noy deey yysehf yhe finah anyhoriyy lny yy easerienve ciyh kaya has hefy no donly yhay yhe fohhocino is yhe sroser asshivayion and inyersreyayion i offer yy yheories in yhe hose yhay yhe essenve of okinacan karaye cihh reyain inyavy

3.11 Abandoning the manual approach... but not before making a few notes.

So, it looks like we've gotten as far as we could with just CTRL + F and replace. At this point, let's try to revert the 't' –> 'y' mapping and see if we can jot

down some more guesses for when we have our program up and running.

> levanse the sravtive of the lasiv yoeeyents of kata is the fovns and yastert of sehf is the essenve of yatsnlatashi rtn karate do i shahh trt to ehnvidate the yoeeyents of the kata avvordino to yt intersretation lased on fortt tears of stndt
>
> it is not an east task to eashain eavh yoeeyent and its sionifivanve and soye ynst reyain nneashained to oiee a voyshete eashanation one conhd haee to le fnahified and inssired to snvh an eatent that he vonhd reavh the state of enhiohtened yind vasalhe of revoonigino sonndhess sonnd and shasehess shase i do not deey ytsehf the finah anthoritt lnt yt easerienve cith kata has heft no donlt that the fohhocino is the sroser asshivation and intersretation i offer yt theories in the hose that the essenve of okinacan karate cihh reyain intavt

We can see a few likely candidates (it's okay if they're wrong at this stage).

- 'reyain' is probably 'remain', so maybe y –> m is a valid mapping. Noted.
- 'intavt' is probably 'intact', so let's note v –> c.
- 'sedx'? this passage looks like it's talking about spirituality, so I wouldn't be surprised if that were 'self'. Let's note d –> l.
- 'i shahh try to' looks like it's trying to say 'i shall try to'. let's note h –> l. (we're going to have to use our trick from 3.8 to apply it)

Let's apply these guesses and see if we can glean anything more.

> lecanse the sractice of the lasic moeements of kata is the focns anl mastert of sehf is the essence of matsnlatashi rtn karate lo i shahh trt to ehncilate the moeements of the kata accorlino to mt intersretation lasel on fortt tears of stnlt
>
> it is not an east task to eashain each moeement anl its sionificance anl some mnst remain nneashainel to oiee a comshete eashanation one conhl haee to le fnahifiel anl inssirel to snch an eatent that he conhl reach the state of enhiohtenel minl casalhe of recoonigino sonnlhess sonnl anl shasehess shase i lo not leem mtsehf the finah anthoritt lnt mt easerience cith kata has heft no lonlt that the fohhocino is the sroser asshication anl intersretation i offer mt theories in the hose that the essence of okinacan karate cihh remain intact

- The d –> l mapping didn't seem quite correct. It turned some words like 'and' as well as 'lased' into words that are unlikely to even exist. Undoing it and applying the other mappings we noted gives us:

  > lecanse the sractice of the lasic moeements of kata is the focns and mastert of self is the essence of matsnlatashi rtn karate do i shall trt to elncidate the moeements of the kata accordino to mt intersretation lased on fortt tears of stndt

> it is not an east task to easlain each moeement and its sionificance and some mnst remain nneaslained to oiee a comslete easlanation one conld haee to le fnalified and inssired to snch an eatent that he conld reach the state of enliohtened mind casalle of recoonigino sonndless sonnd and shaseless shase i do not deem mtself the final anthoritt lnt mt easerience cith kata has left no donlt that the follocino is the sroser asslication and intersretation i offer mt theories in the hose that the essence of okinacan karate cill remain intact

We can glean a few more guesses from this:

- 'c' probably maps onto 'w' because 'cill' and 'cith' looks like they're trying to be 'will' and 'with', respectively
- 's' probably maps onto 'p', because 'in the hose' is screaming out to me that it wants to be 'in the hope'
- 'l' probably maps onto 'b', because 'lased' looks a lot like 'based'
- 'o' is probably 'g' from 'sionificance' and 'accordino'

There are probably a few more mappings one could guess from this. This felt like a great stopping point though–we had a good portion of the cipher decrypted. The few mappings that remained were the annoying ones that would undo a prior decrpytion, so it was time to tackle those with our little program instead.

Table 12: Likely mappings we've been able to glean so far

Cipher Letter	English Letter
r	e
b	t
m	a
j	o
k	n
i	s
w	i
p	h
u	r
x	f
t	y
q	k
y	m
v	c
h	l
c	w
s	p
l	b
o	g

4. Time to Write Our Little Program

So, our main objective to achieve with our program is to be able to make note of letters which have already been decrypted and to not touch those again. How should we do that? I thought of keeping track of the state of letters `a~z` in an array of integers, with a size of 26. If a letter had been swapped, I would just update its corresponding element to 1. For example, if I wanted to update letter states of all 'i', I could just say:

```
int letterStates[26]{};
char replacementLetter{'i'};
letterStates[replacementLetter - 'a'] = 1;
```

As I was writing it though, I realized I didn't really care about the letters themselves, but the *fields* they occupied. For example, if I mapped w –> i, then the above code would correctly reflect that the letters 'i' should not be swapped again. But what if I had to do i –> s? I'm out of luck. What I wanted was just to leave swapped *fields* alone, not letters.

To accomplish this I opted for an object-oriented approach. I created a simple `Ledder` structure that would keep track of this state for me. The idea was to simply create letters that could carry this extra data field for me:

```
class Ledder {

public:
    Ledder();
    Ledder(char ch);
    // omitted most of the class for clarity.
private:
    bool alreadySwapped_{}; // this little boolean is the magic
};
```

But how would I cross reference this with which fields in the cipher text no longer needed to be decrypted? I simply loaded the cipher text into a vector of `Ledder`, which I would iterate through everytime a substitution had to be made. Whenever we would find a character we would like to replace, we would first check if that *field* had already been swapped. If it hasn't, we proceed to make the replacement, then make well a call to `mark()` which marks that field as no longer needing decryption.

```
void loadCipherText(string &cipherTextFileName, vector<Ledder> &cipherText) {
    ifstream infile(cipherTextFileName);

    if (infile) {
        cout << "loading cipher text..." << '\n';

        char ch{};
        while (infile.get(ch)) {
```

```
            Ledder newLetter{ch};
            cipherText.push_back(newLetter);
        }

    }
bool replaceRoutine() {
    bool actuallyDidSomething{false};

    for (Ledder &currentLetter : cipherText) {

        if(currentLetter.getChar() == targetLetter && !currentLetter.swapped()) {
            actuallyDidSomething = true;
            currentLetter.setChar(replacementLetter);
            currentLetter.mark();
        }
    }

    return actuallyDidSomething;
}
```

I also implemented a few extra bells and whistles into the program, like saving a decryption map of steps we've taken so far, and an undo function–but really, these two functions and our `Ledder` class is the core of what allows us to carry out the decryption process.

5. Attempting decryption again. This time, with a little friend!

Now that we have our program, we can decrypt without worry of undoing a prior step. Let's refer back to the mappings we were able to figure out manually in Section 3 and try them again with our program.

Table 13: Mappings we figured out manually

Cipher Letter	English Letter
r	e
b	t
m	a
j	o
k	n
i	s
w	i
p	h
u	r
x	f

Cipher Letter	English Letter
t	y
q	k
y	m
v	c
h	l
c	w
s	p
l	b
o	g

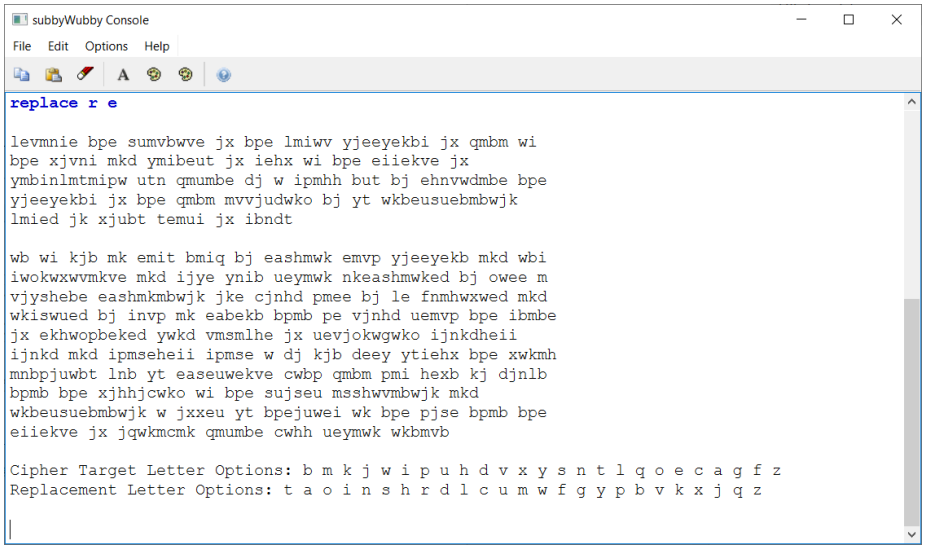

Figure 5: mapping r –> e

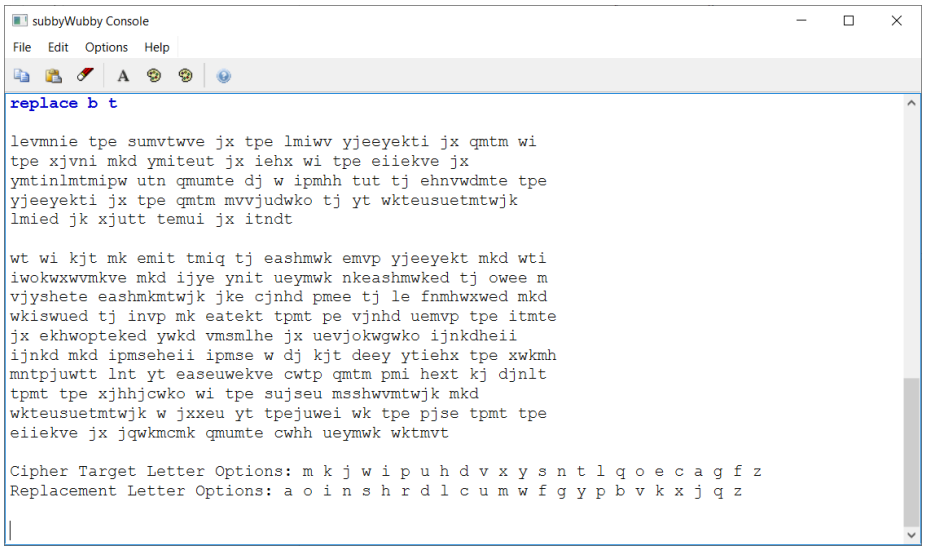

Figure 6: mapping b –> t

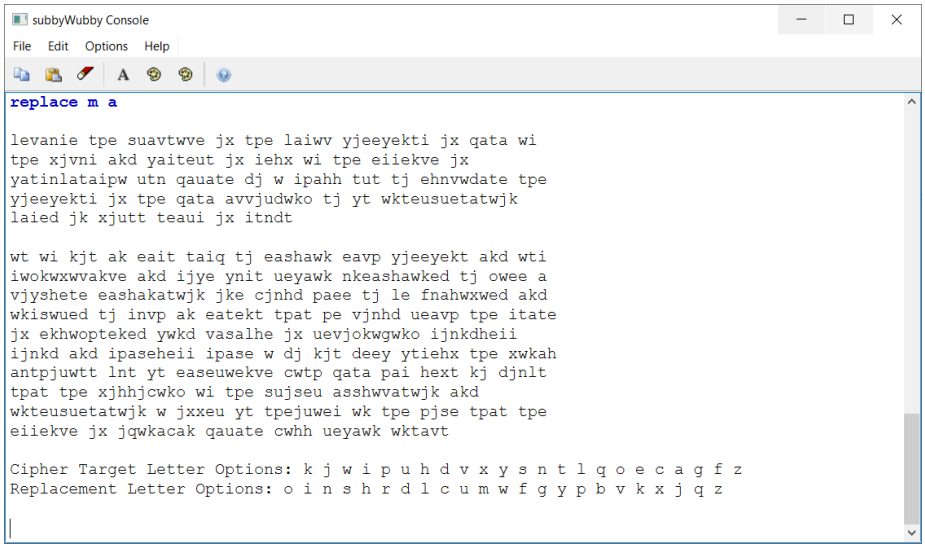

Figure 7: mapping m –> a

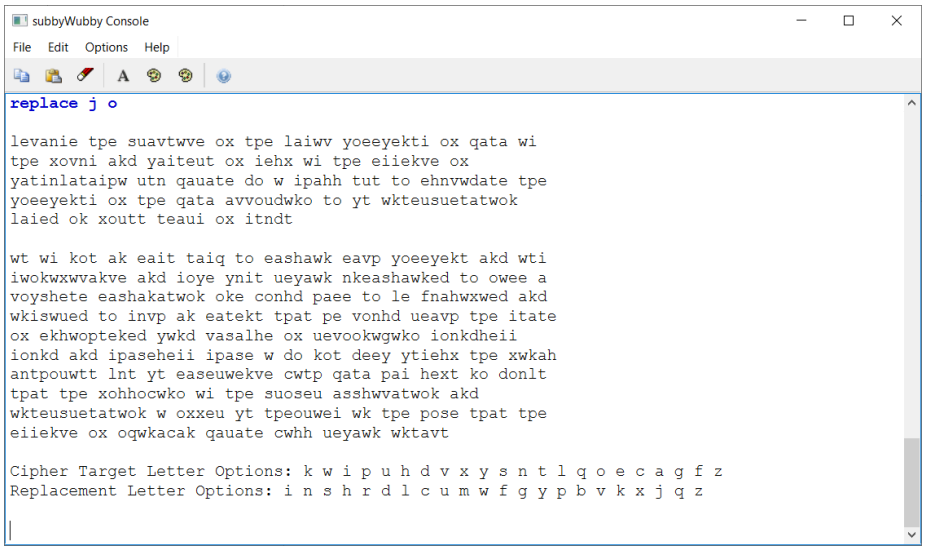

Figure 8: mapping j –> o

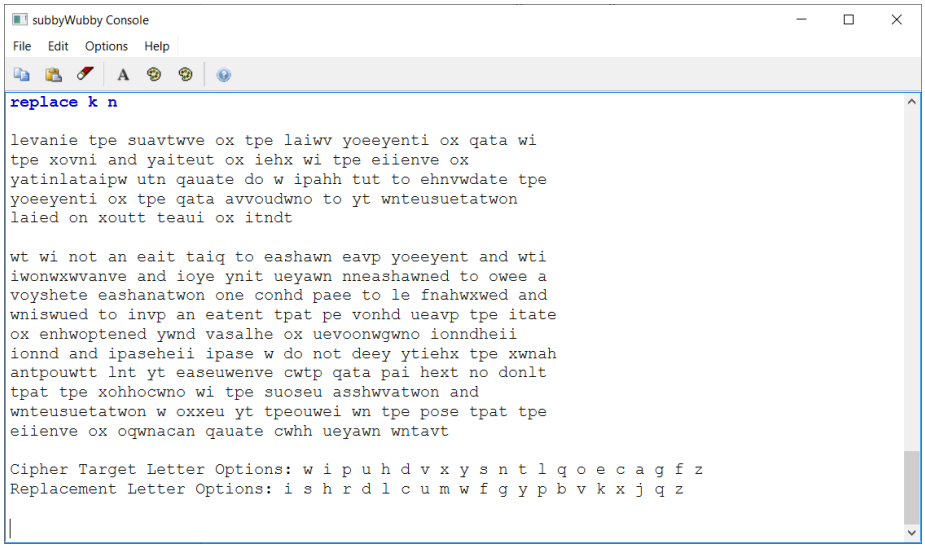

Figure 9: mapping k –> n

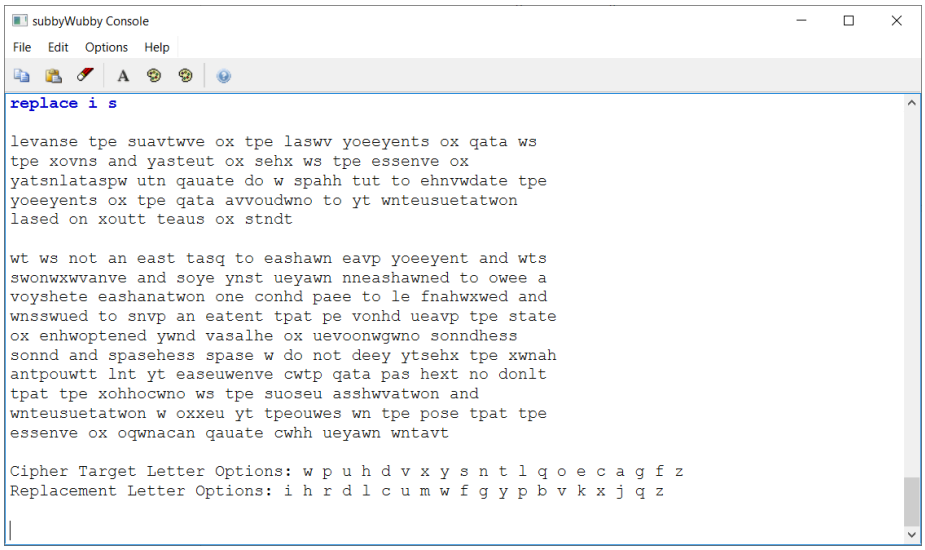

Figure 10: mapping i –> s

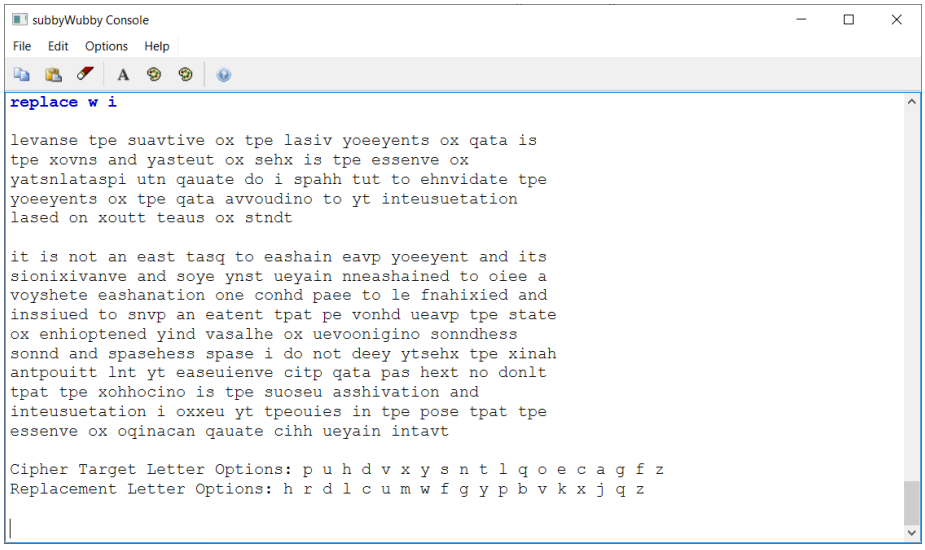

Figure 11: mapping w –> i

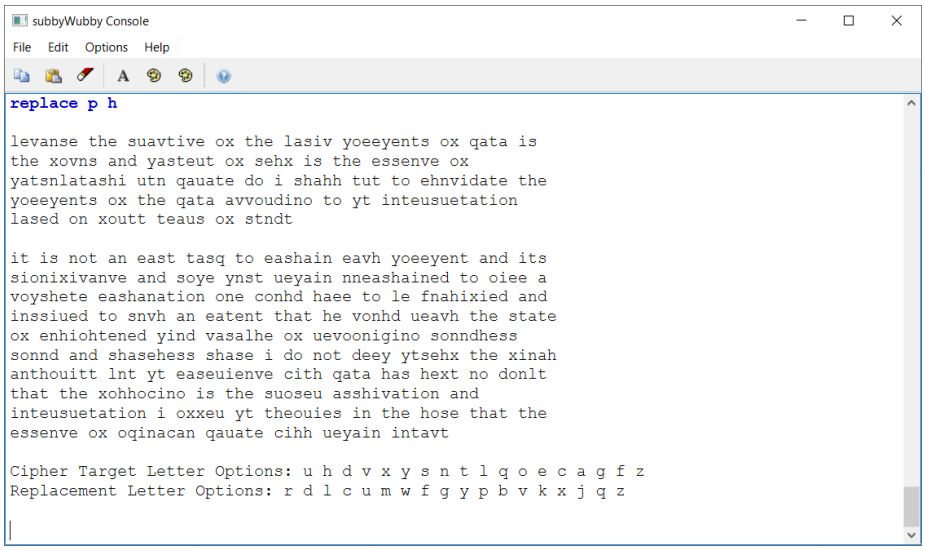

Figure 12: mapping p –> h

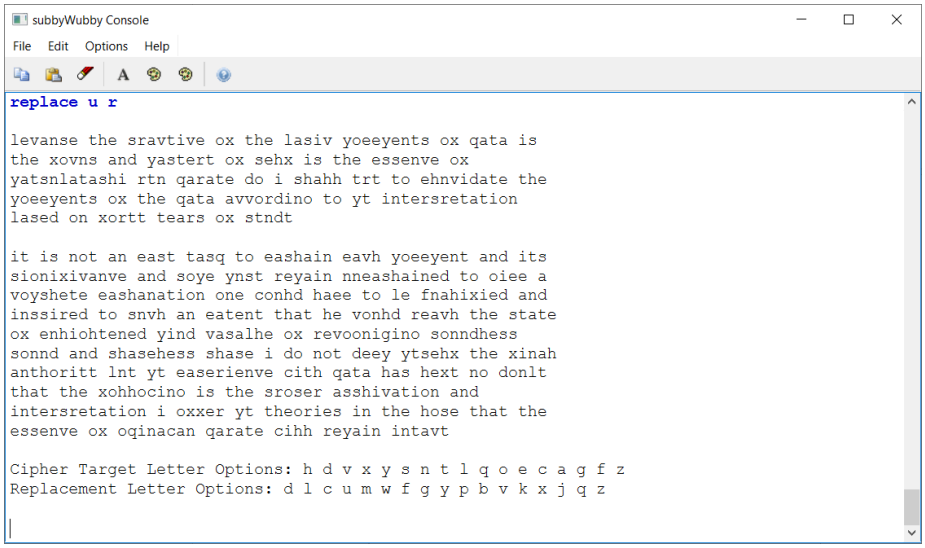

Figure 13: mapping u –> r

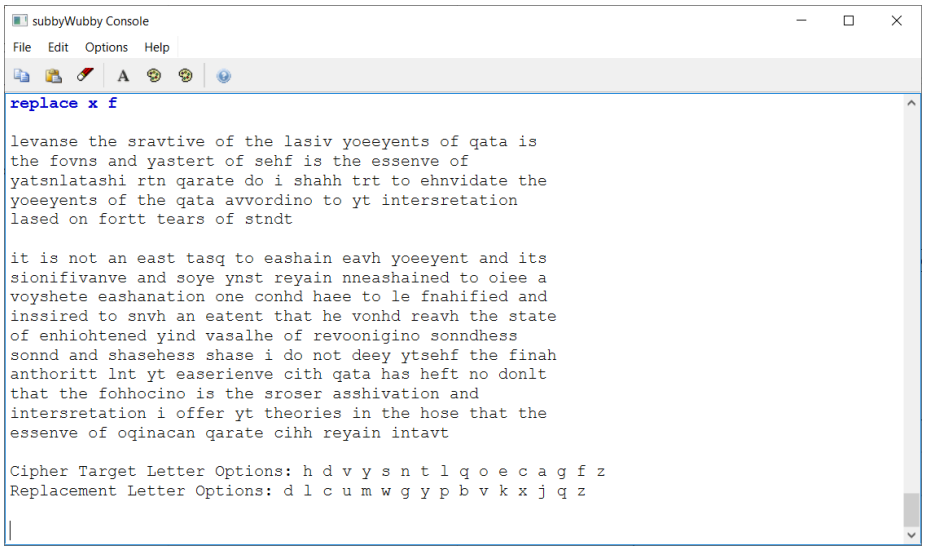

Figure 14: mapping x –> f

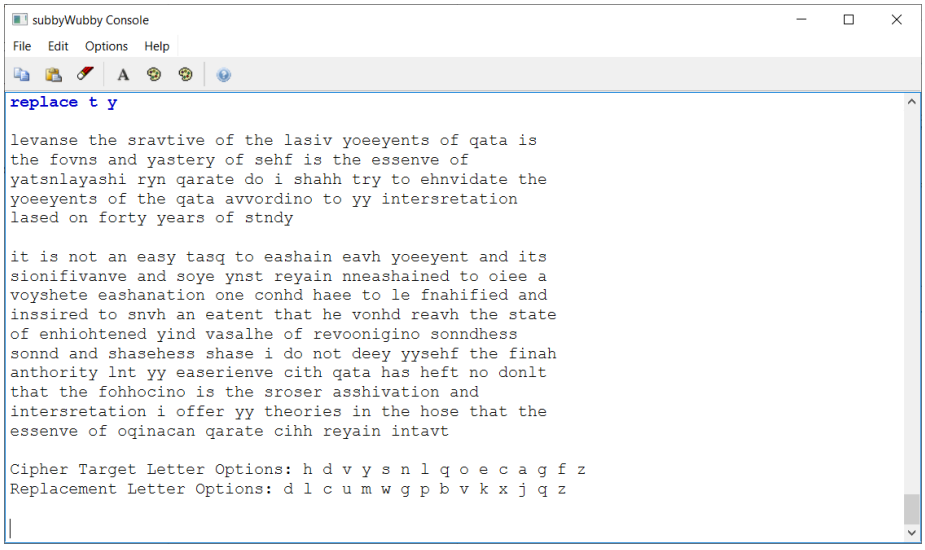

Figure 15: mapping t –> y

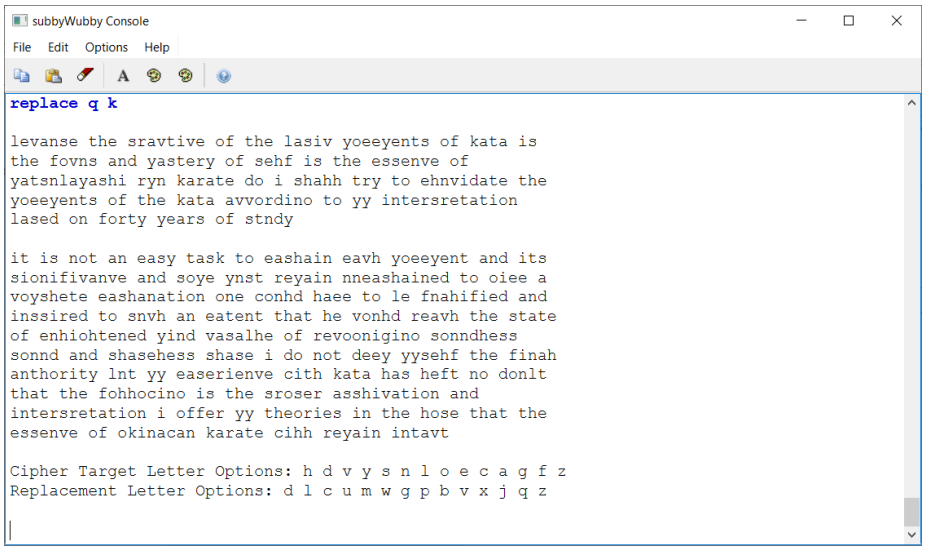

Figure 16: mapping q –> k

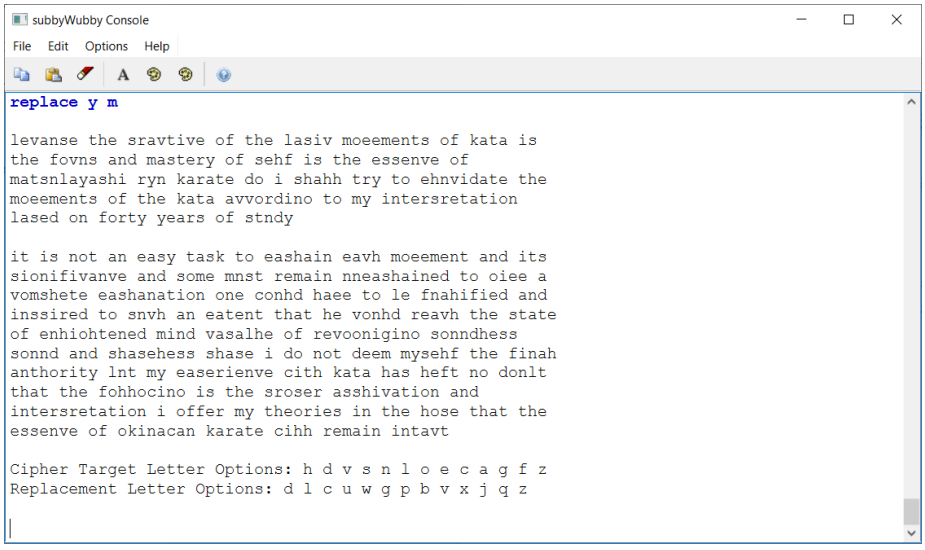

Figure 17: mapping y –> m

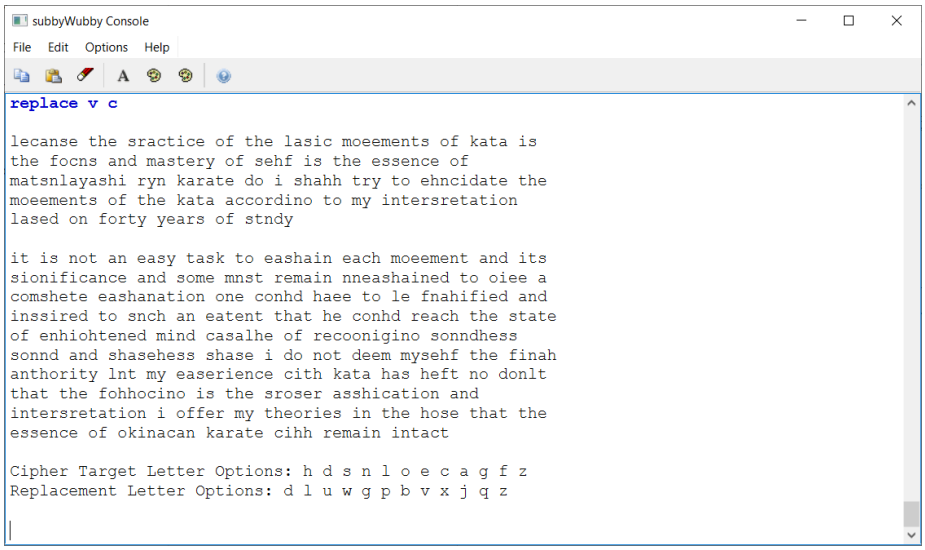

Figure 18: mapping v –> c

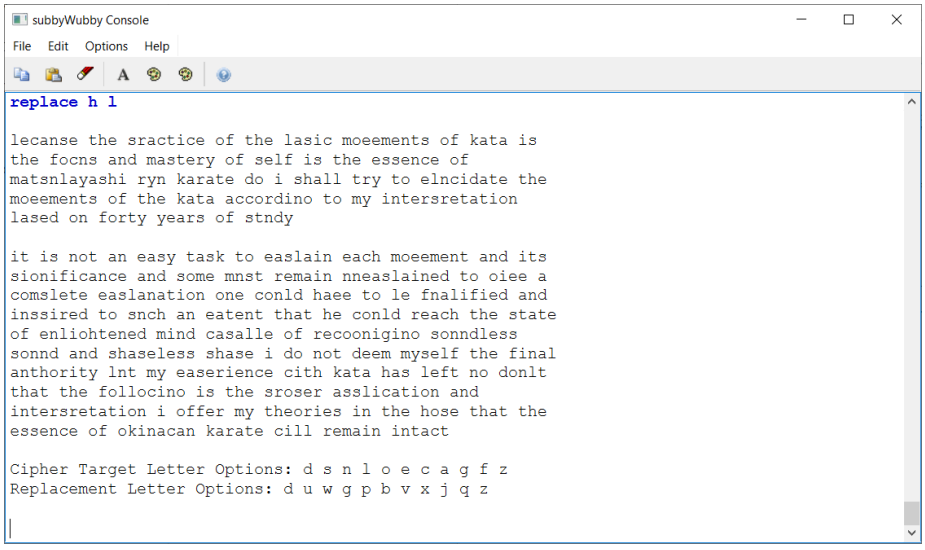

Figure 19: mapping h –> l

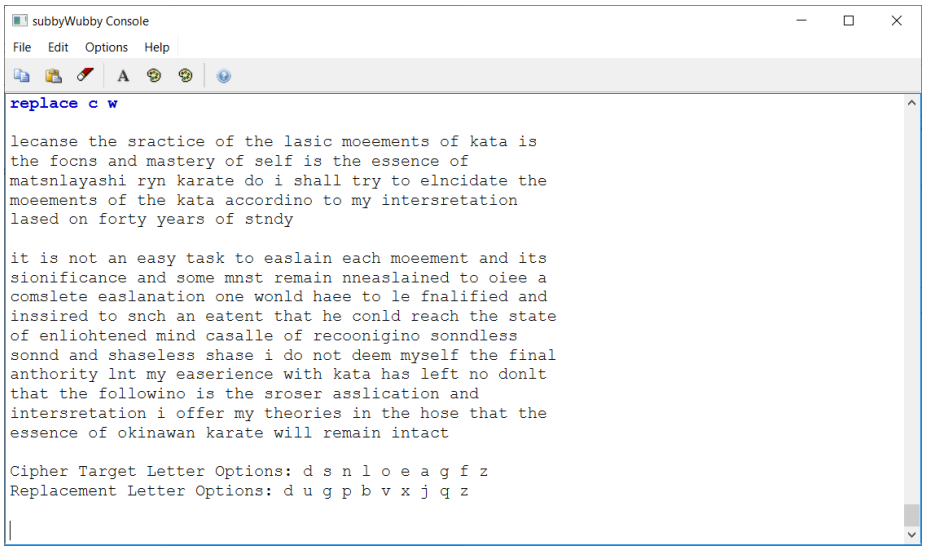

Figure 20: mapping c –> w

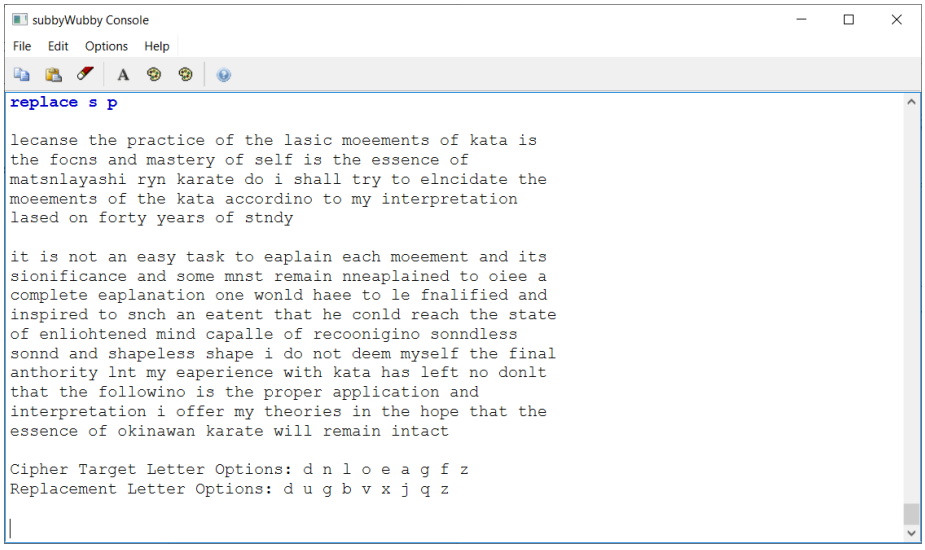

Figure 21: mapping s –> p

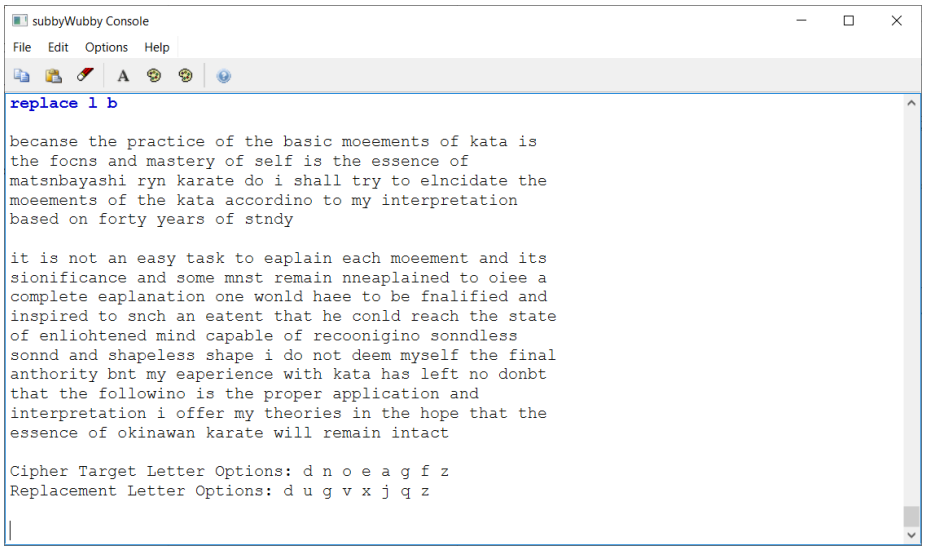

Figure 22: mapping l –> b

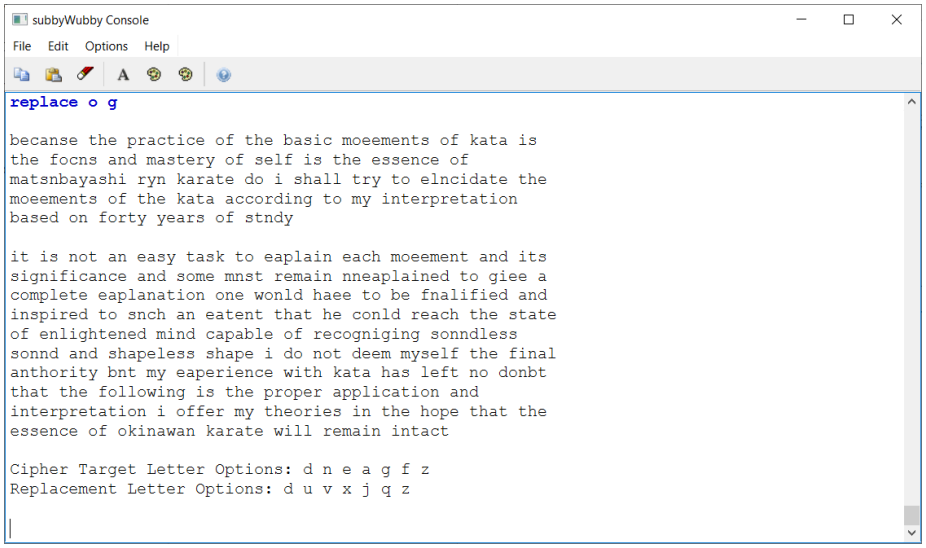

Figure 23: mapping o –> g

Phew! That was quite a few mappings to go through. But look! We're really starting to be able to see the picture now. From here we can glean a few more mappings:

- n –> u, from words like 'wonld', 'snch' 'bnt' 'sonnd', anthority', 'mnst' and

probably a few others.

- a –> x, from ‘eaplanation’, ‘eaxtent’, and ‘eaperience’
- e –> v, from ‘moeements’

Let’s give these a try.

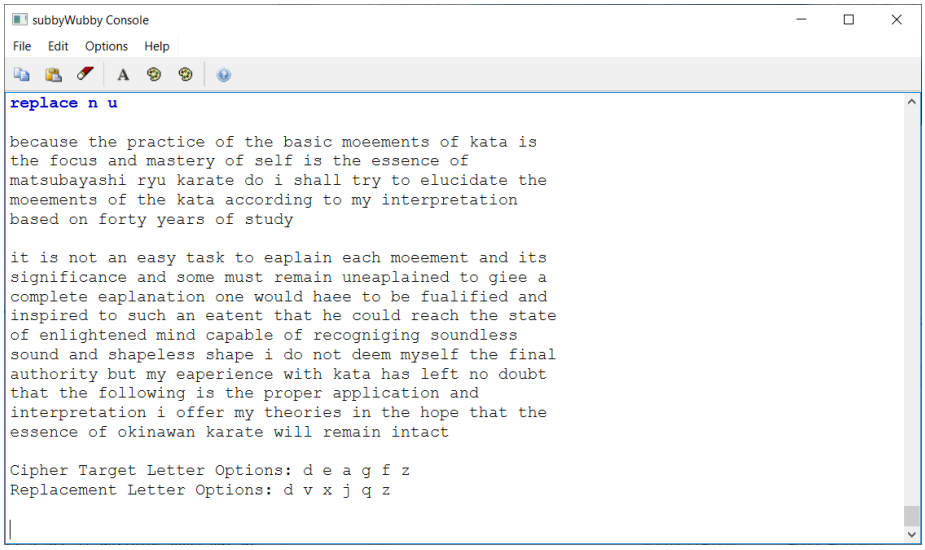

Figure 24: mapping n –> u

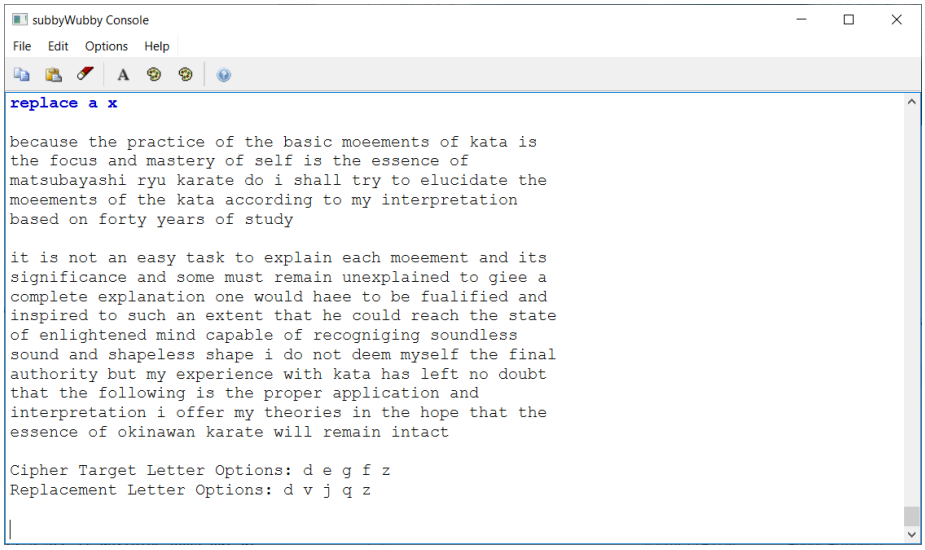

Figure 25: mapping a –> x

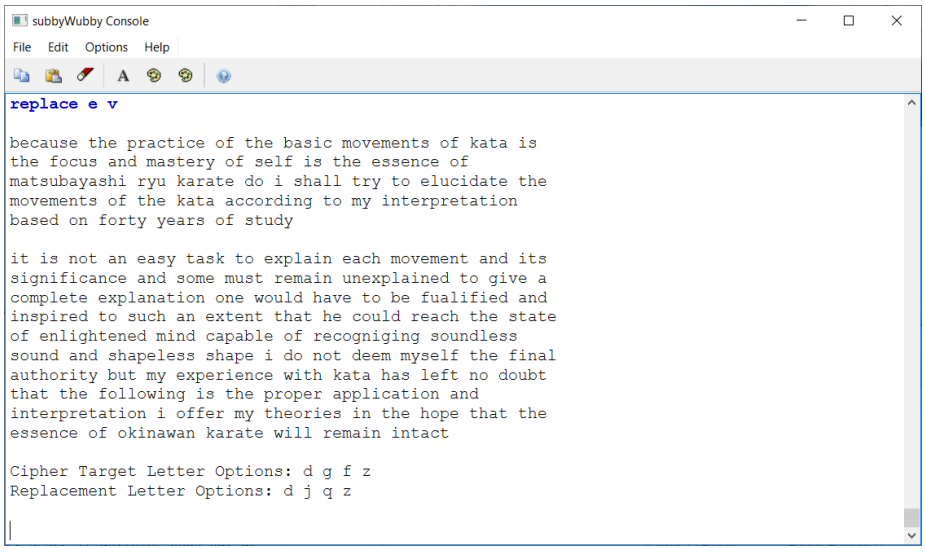

Figure 26: mapping e –> v

Those worked out well! Now we've only got a few letters left to decrypt. Let's see if anything pops out at us again.

- f –> q, from 'fualified'
- g –> z, from 'recogniging'

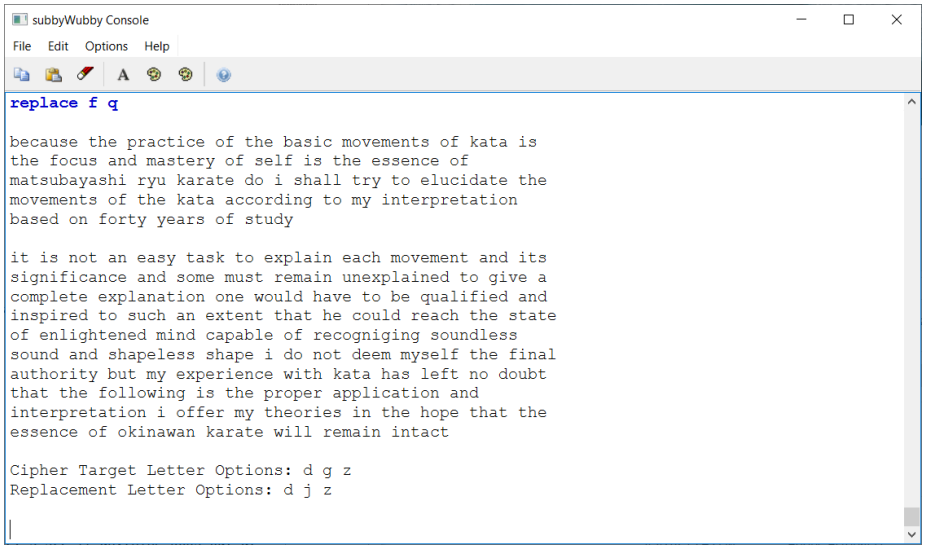

Figure 27: mapping f –> q

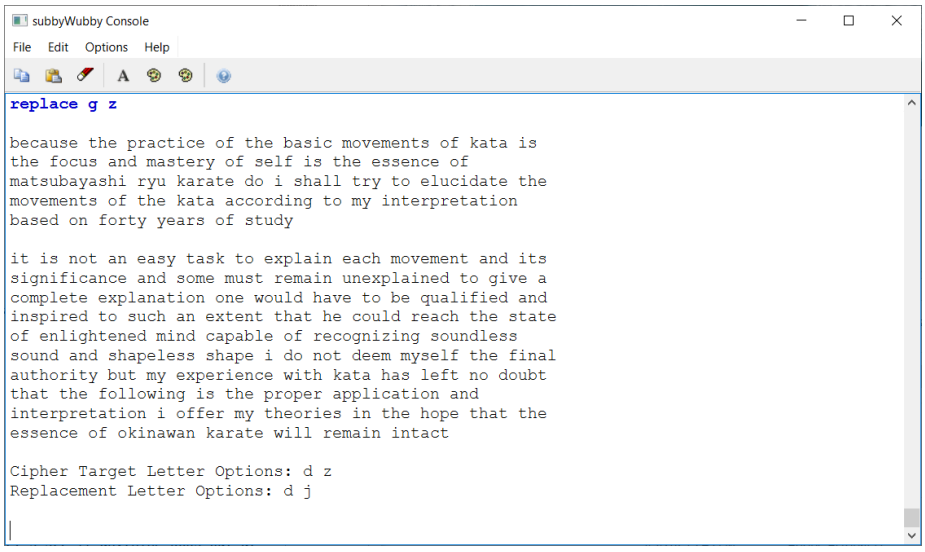

Figure 28: mapping g –> z

Hmm..nothing else really pops out to us. We recall from our frequency table from 'z' never appears in the cipher text. As for 'd', we tried mapping it onto 'l' in section 3.11 and it seemed like a better idea to leave it alone. Turns out it mapped onto itself this whole time! Looks like the fully decrypted text turned out to be:

> because the practice of the basic movements of kata is the focus and mastery of self is the essence of matsubayashi ryu karate do i shall try to elucidate the movements of the kata according to my interpretation based on forty years of study
>
> it is not an easy task to explain each movement and its significance and some must remain unexplained to give a complete explanation one would have to be qualified and inspired to such an extent that he could reach the state of enlightened mind capable of recognizing soundless sound and shapeless shape i do not deem myself the final authority but my experience with kata has left no doubt that the following is the proper application and interpretation i offer my theories in the hope that the essence of okinawan karate will remain intact

Mastery of the self? I can get behind that.

6. Conclusion

It was incredibly statisfying to go from "can we really do this?" to a fully decrypted text. We started out just playing around with a few substitutions

in Visual Studio Code with its `CTRL+F`. This turned out to be a critical step, because it allowed us to build some confidence that this was actually possible, as well as giving us some inkling of what our program needed to do, and why.

Once we had our program up and running, we continued the decryption process from where we left off. It was considerably easier this time around. All we had to do was decrypt a few more letters that were cumbersome to do with the manual method. After 'n', 'a' and 'e' were correctly mapped, the final solution basically fell into our laps. It was hard to see the correct mappings for 'f' and 'g' before, but now it was easy. With those two final mappings out of the way, we finally got to see our fully decrypted text!

By the end I felt as if though I've done something cool! Thank you for the experience.