COM2108 – Autumn Functional Programming Assignment
Thomas Pearson

The Enigma Machine Design

First Design Document

Upon reading the documentation on encoding the enigma simulation, I created Figure 1 (below), a simplified flow chart to visualise the operation of the enigma machine.

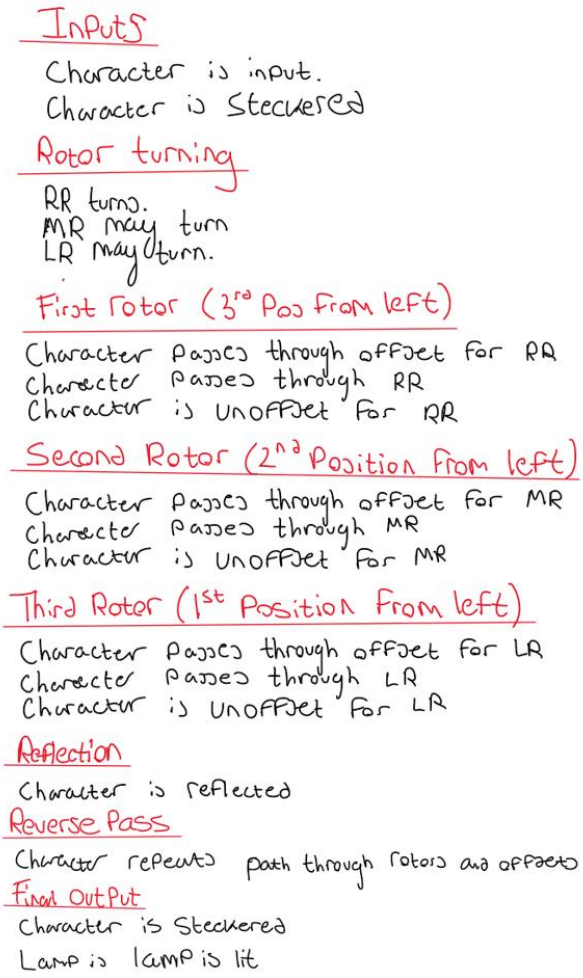

Figure 1 – Design First Draft

Rotor Turning

In this implementation the offset of the rotors is shifted after a character is pressed, but before it is ciphered (as shown in Figure 1 – Design First Draft). The knock-on offset position is also triggered upon reaching it. For example, if the knock-on position was position 17, the knock-on event (turning the connecting rotor) would be triggered upon the rotation from position 16 to position 17. However, in this implementation the characters are offset not the actual rotors. This is discussed further below.

Character Offsets

As demonstrated by Figure 1 – Design First Draft, instead of turning the rotors, which would be very resource intensive, I have decided to offset the input by the rotor amount. This way it is easier to keep track of the position of the current rotor and therefore the needed offset. It also ensures that the connecting rotor receives the correct character when the letter is passed to the next rotor.

For example, if the "Rotor I" was offset by three positions and the character "A" was input it would process the information as follows:

1) Shift "A" along the alphabet by three position to get "D".
2) Pass letter "D" through the rotor to get "F"
3) Shift "F" backwards by three positions through the alphabet to get character "C"
4) Pass character "C" on to the next rotor and repeat the same process

This approach simulates the rotor having shifted three positions physically as the connecting rotor is unaware of the shifted previous rotor. It is only away of the position of the incoming signal in its own rotor. If we were to not "unshift" the encrypted value it would be receiving the incorrect value from the previous rotor.

Second Design Iteration

Once the main parts of the design were created it was divided further into smaller functions. The flow of these proposed functions is shown in Figures 2 and 3.

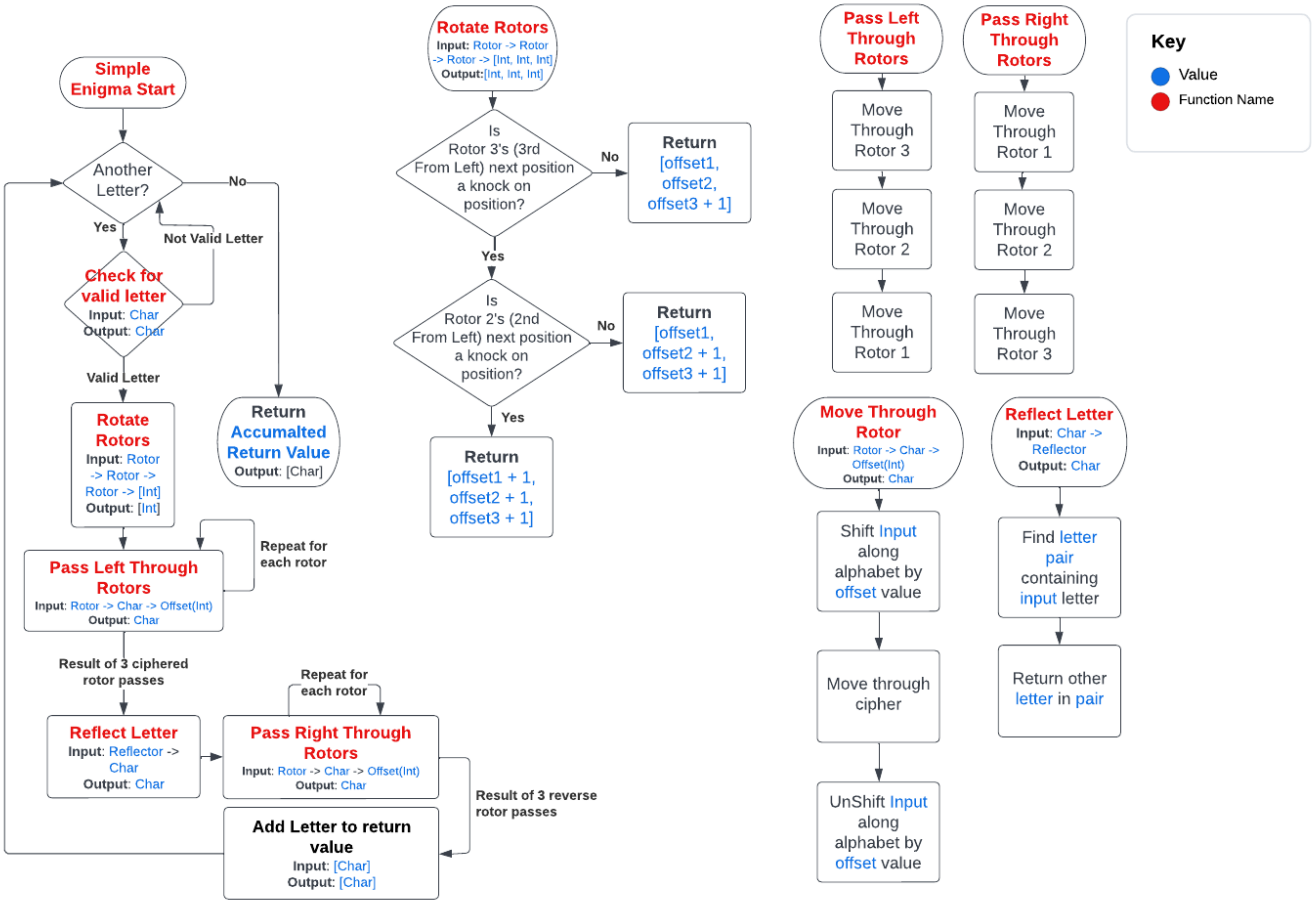

Figure 2 - Simple Enigma Operation

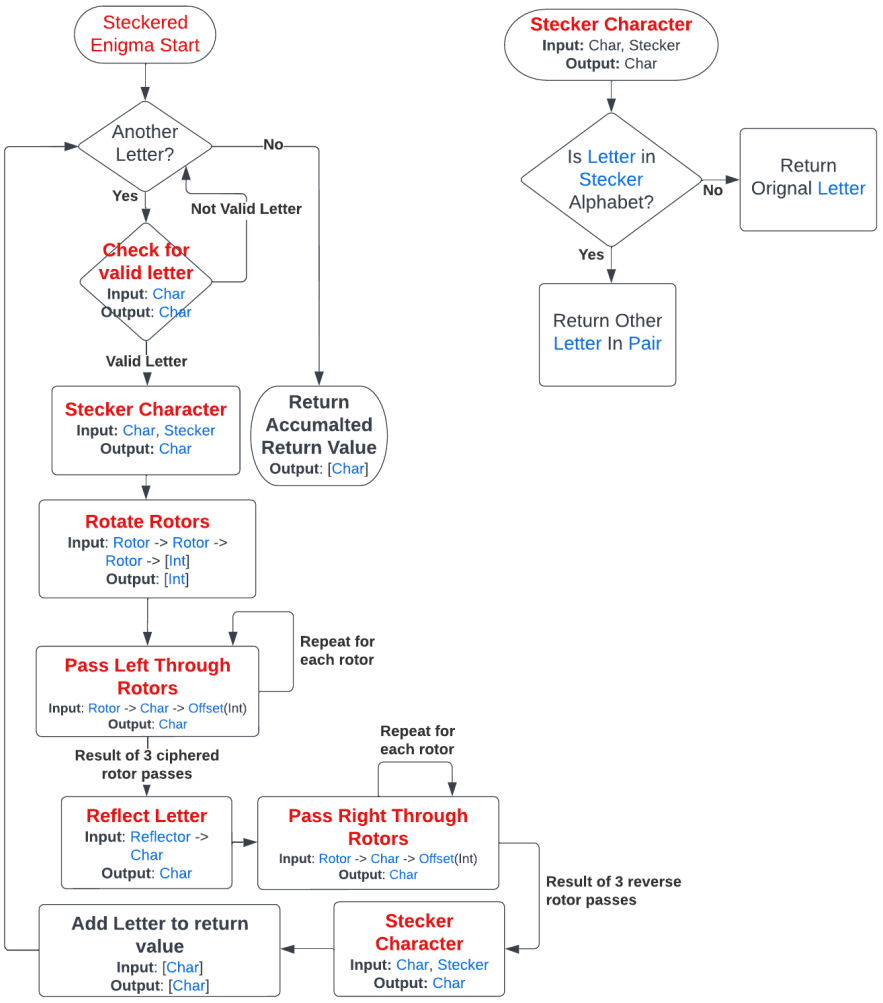

Figure 3 - Steckered Enigma Operation

Enigma Machine Testing

The first functions that were implemented were the ones that had the least dependencies on other functions. The tables below show the order the functions were implemented.

Reflector Function

Named “reflectorFunction” in the implementation, it simply takes in a character and list of reflector pairs to find the input character’s paired letter.

Test Input (Character, Reflector)	Rational Behind Values	Expected	Output
‘A’, reflectorB	Test for values input on the left of the pair are reflected	‘Y’	‘Y’
‘B’, reflectorB	Test for element other than first	‘R’	‘R’
‘W’, reflectorB	Test for values input on the right of the pair reflected.	‘V’	‘V’

Rotate Rotors

Named “rotateRotors” in the implementation. It takes in three separate rotors and an array of their current offsets. If any rotor’s next position is a knock-on position the next rotor in the series will turn. If any rotor’s increment hits 26 it will return to position 0 (rotor has returned to starting position).

Test Input (Rotor, Rotor, Rotor, Offsets)	Rational Behind Values	Expected	Output
rotor1, rotor2, rotor3, [0,0,0]	First shift for starting input	[0,0,1]	[0,0,1]
rotor1, rotor2, rotor3, [0,1,25]	Checks for return to starting value	[0,1,0]	[0,1,0]
rotor1, rotor2, rotor3 [0,0,21]	Checks for knock on position for third rotor	[0,1,22]	[0,1,22]
rotor1, rotor2, rotor3 [0,4,21]	Checks for dual turning of rotors	[1,5,22]	[1,5,22]
rotor1, rotor2, rotor3 [16,4,21]	Ensures first rotor has no impact on other rotors	[17,5,22]	[17,5,22]

Integer to Letter Conversion

Named “int2let” in code. It performs the reverse of the “alphaPos” provided function and converts a letter index into its equivalent letter.

Test Input (Int)	Rational Behind Values	Expected	Output
0	Check function converts indexes correctly	‘A’	‘A’
25	Correct last letter of alphabet	‘Z’	‘Z’
13	Random check	‘N’	‘N’

Shift Letter in the Alphabet

Named “shiftInput” in the code. It moves the letter input forwards in the alphabet by the specified input number of spaces. It will loop back around to the start of the alphabet if the input goes past ‘Z’ value.

Test Input (Int, Character)	Rational Behind Values	Expected	Output
0, ‘A’	Base case	‘A’	‘A’
1, ‘Z’	Loop back to first letter of alphabet	‘A’	‘A’
25, ‘Z’	Test of function	‘Y’	‘Y’
4, ‘B’	Test of function	‘F’	‘F’
30, ‘A’	Test if too large of input entered	‘E’	‘E’

Unshift Letter in the Alphabet

Named “unShiftInput” in the code. This function performs the opposite to “shiftInput” and moves a letter backwards along the alphabet by the specified number of positions. It also loops back to the end of the alphabet if the letter goes past the start of the alphabet.

Test Input (Int, Character)	Rational Behind Values	Expected	Output
0, 'A'	Base Case	'A'	'A'
1, 'A'	Loop back to end of the alphabet	'Z'	'Z'
5, 'H'	Test functionality	'C'	'C'
30, 'A'	Test if too large of input entered	'W'	'W'

Find Passed Through Value

Named "findLetterPosition" in the code. This function takes an array of characters, a character to look for and a counter which by default should be 0. Its purpose it to look for the position of the character in the provided array. This is useful when we passing a character back through a rotor. We have the current value that is being passed back but not its position. This function finds it.

Test Input ([Char], Char, Int)	Rational Behind Values	Expected	Output
"EKMFLGDQVZNTOWYHXUSPAIBRCJ", 'E', 0	Base case	0	0
"EKMFLGDQVZNTOWYHXUSPAIBRCJ", 'J', 0	End of character array	25	25
"EKMFLGDQVZNTOWYHXUSPAIBRCJ", 'A', 20	Functionality Checker	20	20

Pass Left Through Rotors

Named "passLeft" in the code. This function takes a rotor, a character to cipher and the offset of the rotor. First it shifts the input character by a desired number of positions using "shiftInput". Then it finds the index of the character in the alphabet to find the index of the equivalent value in the rotor. Finally, it unshifts the result by the number of places provided using "unShiftInput".

Test Input (Rotor, Character, Int)	Rational Behind Values	Expected	Output
Rotor1, 'A', 0	Base case test	'E'	'E'
Rotor2, 'A', 0	Base case test	'A'	'A'
Rotor1, 'A', 1	Offset increased	'J'	'J'
Rotor1, 'B', 1	Offset increased and character changed	'L'	'L'
Rotor1, 'A', 26	Checks that result loops back around if offset exceeds alphabet length	'E'	'E'
Rotor2, 'Z', 30	Checks that result loops back around if offset exceeds alphabet length	'B'	'B'

Pass Right Through Rotors

Named "passRight" in the code. This function's operation is similar to pass left except that instead of passing in the letter's index in the alphabet it uses the letter's index in the rotor. For example, in rotor 1 its alphabet is "EKMFLGDQVZNTOWYHXUSPAIBRCJ", the letter M is at index 2 (starting at 0). This is neccesary to pass the letter backwards through the rotor.

Test Input (Rotor, Character, Int)	Rational Behind Values	Expected	Output
Rotor1, 'A', 0	Base case test	'U'	'U'
Rotor2, 'A', 0	Base case test	'A'	'A'
Rotor1, 'A', 1	Offset increased	'V'	'V'

Rotor1, ‘B’, 1	Offset increased and character changed	‘X’	‘X’
Rotor1, ‘A’, 26	Checks that result loops back around if offset exceeds alphabet length	‘U’	‘U’
Rotor2, ‘Z’, 30	Checks that result loops back around if offset exceeds alphabet length	‘Y’	‘Y’

Stecker Input (Plug Board)

Named “steckerPass” in the code. Its purpose is to stecker the input based on the provided stecker pairs. If the letter is not found in the stecker pairs then it will return the original letter.

Test Input (Char, [(Char, Char)])	Rational Behind Values	Expected	Output
'A', [('F','T'), ('D','U'), ('V','A'), ('K','W'), ('H','Z'), ('I','X')]	Right side value	‘V’	‘V’
'I', [('F','T'), ('D','U'), ('V','A'), ('K','W'), ('H','Z'), ('I','X')]	Left side value	‘X’	‘X’
'A', [('F','T'), ('D','U'), ('V','A'), ('K','W'), ('H','Z'), ('I','X')]	Value not in stecker	‘B’	‘B’

Encode Message

The largest function in the program, it takes in a String to be encoded and the initial setup of the enigma machine. It uses recursion to cipher a letter one at a time according to the flow chart in the design document. Before a letter is ciphered it is checked to be a valid letter. Its symmetric nature is also useful for testing its correct implementation.

To Simplify the layout of the test data please refer to this table as a list of enigma machines that were used:

- Enigma1 = (SimpleEnigma rotor1, rotor2, rotor3, reflectorB, (0,0,0))
- Enigma2 = (SimpleEnigma rotor1, rotor2, rotor3, reflectorB, (25,25,21))
- Engima3 = (SimpleEnigma rotor4, rotor5, rotor2, reflectorB, (25,25,21))

Simple Enigma Testing

Test Input (String, Enigma)	Rational Behind Values	Expected	Output
“Test”, Engima1	Basic implementation functionality.	"OLPF"	"OLPF"
"OLPF", Engima1	Reflective nature	“TEST”	“TEST”
“ Test”, Enigma1	Data sanitisation	“OLPF”	“OLPF”
“@@@#123Test][”, Enigma1	Data sanitisation	“OLPF”	“OLPF”
“Test”, Enigma2	Offset rotation	“YWYV”	“YWYV”
“YWYV”, Enigma2	Reflection confirmation	“TEST”	“TEST”
“Test”, Enigma3	Changed rotors	“WLYL”	“WLYL”
“WLYL”, Engima3	Reflection confirmation	“TEST”	“TEST”

Steckered Enigma Testing

- Plugboard = [('F','T'), ('D','U'), ('V','A'), ('K','W'), ('H','Z'), ('I','X')]
- Enigma4 = (SteckeredEnigma rotor1 rotor2 rotor3 reflectorB (0,0,0) plugboard)
- Enigma5 = (SteckeredEnigma rotor1 rotor2 rotor3 reflectorB (25,25,21) plugboard)
- Enigma6 = (SteckeredEnigma rotor1 rotor5 rotor4 reflectorB (25,25,21) plugboard)

Test Input (String, Enigma)	Rational Behind Values	Expected	Output
“Test”, Enigma4	Basic implementation functionality checker	“ELPF”	“ELPF”
“ELPF”, Enigma4	Reflection confirmation	“TEST”	“TEST”
“@@@#’’#’#Test##”, Enigma4	Data Sanitisation	“ELPF”	“ELPF”
“Test”, Enigma5	Offset rotation	“ZKYM”	“ZKYM”
“ZKYM”, Enigma5	Reflection confirmation	“TEST”	“TEST”
“Test”, Enigma6	Changed rotors	“RPLW”	“RPLW”
“RPLW”, Enigma6	Reflection confirmation	“TEST”	“TEST”

Example Data Test

On blackboard one set of test data was provided using the Enigma (SimpleEnigma rotor3 rotor2 rotor1 reflectorB (0,0,25)). According to my design documentation the sample that is applicable to my system is sample 2.

Test Data	Result
"NIQVDYIAJGLEOWOAPVFKXAFXNXXSBCNXTCBEBWJQDULHWRG AORIUVFQXCNYORCZIJZRWNUASAFNXAPOXOJQWUCVBCDJGLTX UZZEDPBEZKSRUUFDGTWNAGQUKNUYXHJUTDDOVLJDUVZBRRZ HLENTSHCSBQVAAVQBSGXDNWGDQGZXRIUYMPEERAKDWDIUTL OXIVXAJPNYEXOQLHLECWKXAUXOPFWCKMMHDAXJALRENHIOB MREPOZIPOEUZSTCOYBCDGVNYSZKPAIAONPTBCZEZTZOTTKLDRJ PLPUIPTJVZWSFCTUHRJFRXSXGOGJCXZVOINGGNMFBIFNJHNEUH HSKRVKOSVOURUYDQLZECVZXTMTVSHZNTBURTBSFXTXNEOVGX DKRSEXSXYGGTRFDJLRJWPYUSJZHVLHTESMOLALVLKODNGJKZRJY GOXOFQJPQJZGHZMOVXCFQWRUHZYMWUSXMWOFAXGEKSBM OVARUTGKCJXFICGFMJLTKXTVGOTPCOSHQGBUYTGPUNHAWGQ SXBJHSVHBRHHEPOGRQMSARYFMPTARVHLUQWBGTCJFCEGBBD TIBOIZQQRJUAJMEJJMWFUVYBESQENYXHFEGYQPKOHIBLECZQEN PVNWYOPBVDQGDBKDCSPBFBUPLGJNIPLIDPHMAMHSHWGNYIZ PNHPVDJTQXMDKAJPMWWQEAMBCLDNWOEHSWHRMBGDAMA GOWWNPOOAPVGRYATMHSEOUFPTOJAQZFNBSAFNVWQUXUOZ GQFMUKQYBBVBOVEJNGRPUXZLJTTBQQBEERYDETTXKHRILCUDV ZJLZXQNRKJGLMPMDPFYPDITHGZMBLEKNFWPXLANWSHSXCEDU PYDJGTTUKQJFZOPCQOFZWXIGBWFXSDYQUYRDRLWDIJQODWZJ BGXJHFLTTVLGHKEMSGXKUZLFSHPGRLUVABFXZZCNQLGQZABBJP LIZYONCASBCFNEMTSPPYSHDHZHGLZGWXZECNUHJXOYMJFGJWY JTSAEYPJHRPAGNUMFAREXVBGDYFZUNYQFKXFMVKOVWLYYYAA AMLQSQFQTUNKKSFXJMJJEJFEKQNDMFJVOLOTFCXFHSTEAIOTEF LXFJRSNHZSQKJMUUZICHFOIMODKSDZMMWXZDLWAOPALTCFH GCZGPERIOHWIAKWRAPHEDUAVREGLQWXOULISLRREVNNHQYG LOZYBFZNLRXSFXIOYZPWRDXZNEFFZQMUUJEORUJWVFIEGXLJGK MBIDBSKHMSUDEGWJQJIAACWFUUDITVJYWHIQIEFPEPDAHRPTG MUOCHZBGQFHWZORDUJIWKZJBZWMXYAKNWZVDBWUMGMTV TQAKISPVNDCWZNPKOBMZDREHWAPAADLRMWCZPAHHNZVKX QVQSGTTZHYCXVPMDCDBCLSSRQUWTUKGLJLFPFJRELPGWHCZQ FQEUAGQKBBHJHLBLZXATYGMBFVLUKWRKODYPFMQHXSCLGLCX XXDNILVBBBYQDAPOZFZJPXAMKVTEUOVWEACVSVBCAPTZSRTICE NZGTBBCAKAVHCGOHLBTMHWZVOVOFHHSWFVARJGGEFVGJUXZ JYGQUQPEJCEYTOPDXSIPMLEOIMGJTTIVSGOHLXSYSYSOOORERC QYQHYHITCJUKYXJTCSHYVEXAMOJHKBOUNIGFIESPTSQMWRGYY GIQSLAERWQTIQSUCMLPIUOXSYWXTCAIPQIAEKYADIDAAGLSKZV CCUSCPWPXXHYBGJFGHKEPADEULODUSHPWFHVHFLLPHAMBQT	“ALICEWASBEGINNINGTOGETVERYTIREDOFSITTINGBYHERSISTER ONTHEBANKANDOFHAVINGNOTHINGTODOONCEORTWICESHEHA DPEEPEDINTOTHEBOOKHERSISTERWASREADINGBUTITHADNOPIC TURESORCONVERSATIONSINITANDWHATISTHEUSEOFABOOKTHO UGHTALICEWITHOUTPICTURESORCONVERSATIONSSOSHEWASCO NSIDERINGINHEROWNMINDASWELLASSHECOULDFORTHEHOTDA YMADEHERFEELVERYSLEEPYANDSTUPIDWHETHERTHEPLEASURE OFMAKINGADAISYCHAINWOULDBEWORTHTHETROUBLEOFGETTI NGUPANDPICKINGTHEDAISIESWHENSUDDENLYAWHITERABBITWI THPINKEYESRANCLOSEBYHERTHEREWASNOTHINGSOVERYREMAR KABLEINTHATNORDIDALICETHINKITSOVERYMUCHOUTOFTHEWAY TOHEARTHERABBITSAYTOITSELFOHDEAROHDEARISHALLBELATE WHENSHETHOUGHTITOVERAFTERWARDSITOCCURREDTOHERTHA TSHEOUGHTTOHAVEWONDEREDATTHISBUTATTHETIMEITALLSEE MEDQUITENATURALBUTWHENTHERABBITACTUALLYTOOKAWATC HOUTOFITSWAISTCOATPOCKETANDLOOKEDATITANDTHENHURRI EDONALICESTARTEDTOHERFEETFORITFLASHEDACROSSHERMIND THATSHEHADNEVERBEFORESEENARABBITWITHEITHERAWAISTCO ATPOCKETORAWATCHTOTAKEOUTOFITANDBURNINGWITHCURIO SITYSHERANACROSSTHEFIELDAFTERITANDFORTUNATELYWASJUS TINTIMETOSEEITPOPDOWNALARGERABBITHOLEUNDERTHEHEDG EINANOTHERMOMENTDOWNWENTALICEAFTERITNEVERONCECO NSIDERINGHOWINTHEWORLDSHEWASTOGETOUTAGAINTHERAB BITHOLEWENTSTRAIGHTONLIKEATUNNELFORSOMEWAYANDTHE NDIPPEDSUDDENLYDOWNSOSUDDENLYTHATALICEHADNOTAMO MENTTOTHINKABOUTSTOPPINGHERSELFBEFORESHEFOUNDHERS ELFFALLINGDOWNAVERYDEEPWELLEITHERTHEWELLWASVERYDE EPORSHEFELLVERYSLOWLYFORSHEHADPLENTYOFTIMEASSHEWE NTDOWNTOLOOKABOUTHERANDTOWONDERWHATWASGOINGT OHAPPENNEXTFIRSTSHETRIEDTOLOOKDOWNANDMAKEOUTWHA TSHEWASCOMINGTOBUTITWASTOODARKTOSEEANYTHINGTHENS HELOOKEDATTHESIDESOFTHEWELLANDNOTICEDTHATTHEYWERE FILLEDWITHCUPBOARDSANDBOOKSHELVESHEREANDTHERESHES AWMAPSANDPICTURESHUNGUPONPEGSSHETOOKDOWNAJARFR OMONEOFTHESHELVESASSHEPASSEDITWASLABELLEDORANGEM ARMALADEBUTTOHERGREATDISAPPOINTMENTITWASEMPTYSHE DIDNOTLIKETODROPTHEJARFORFEAROFKILLINGSOMEBODYUNDE

ZXONDTSLVCZKZDBIPOMIXFBLHZFVOICUOFUXLHPLKBYRTDKMAT ETHKDHCXQRDBFCKFGPXNEIRLAPNUAPOQXI"	RNEATHSOMANAGEDTOPUTITINTOONEOFTHECUPBOARDSASSHE FELLPASTIT”

My implementation is able to decrypt the message correctly into the Alice in Wonderland test.

Finding the Longest Menu Design

First Design Iteration

Upon reading the documentation on encoding the enigma simulation, I created Figure 4 (below), a simplified flow chart in order to visualise the operation of the enigma machine.

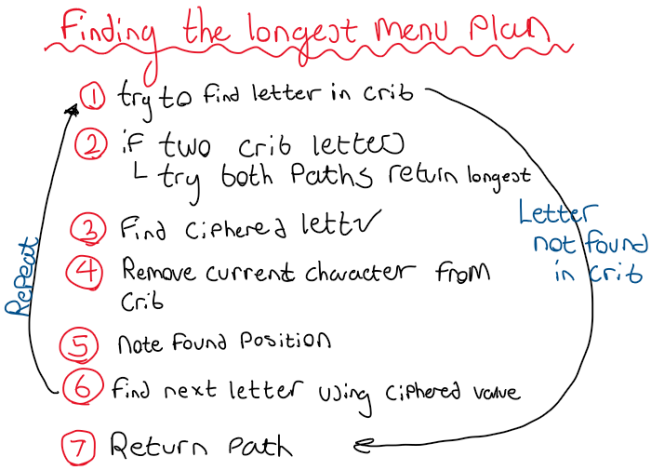

Figure 4 - Initial Plan for Longest Menu

After analysing this initial plan there was an issue with how the program would deal with multiple paths in the route. This is best explained in an example as follows.

There are two paths for the letter “A” in the route, meaning that the program has to calculate two possible paths. Further down in the computation, both paths encounter two possible paths for the letter "B" and the program attempts to find the best route. There are now 4 paths being computed. This logic problem very quickly gets quite large and, in the current design, is recomputed after the longest path is found as all it does, after finding the best route for "A" (and therefore the best route for all options) is return the next index, not the next few steps. Therefore, when the program computes the best path for “A”, it needs to compute the best “B” path although it has previously already computed it.

A simple solution to this program is that instead of just returning the next index when two paths are found but the remaining path. Therefore, it does not recompute the problem. This approach is also applicable to finding the smaller problems in the program, reducing the overall computation time.

Second Design Iteration

The second design iteration focused on clarifying the operation of the original design and implementing the new path return feature.

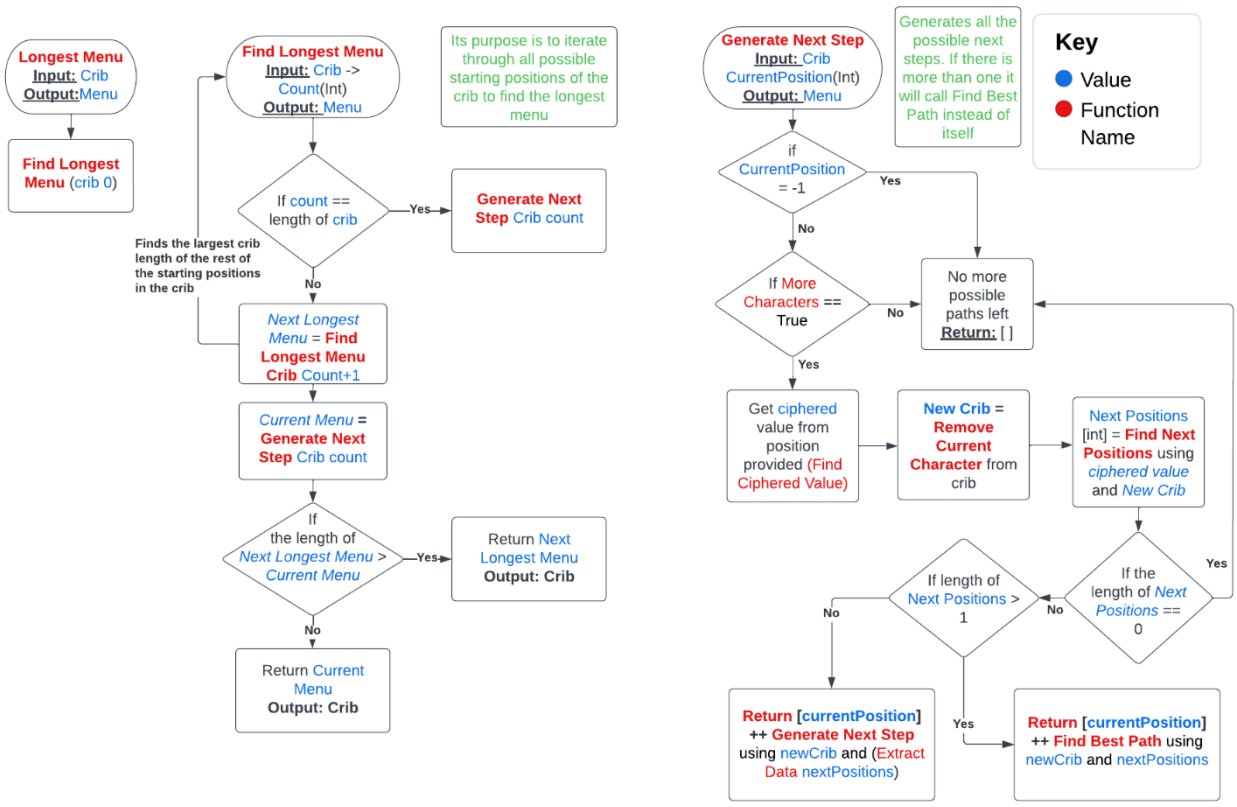

Figure 5 - Longest Menu Design Part 1

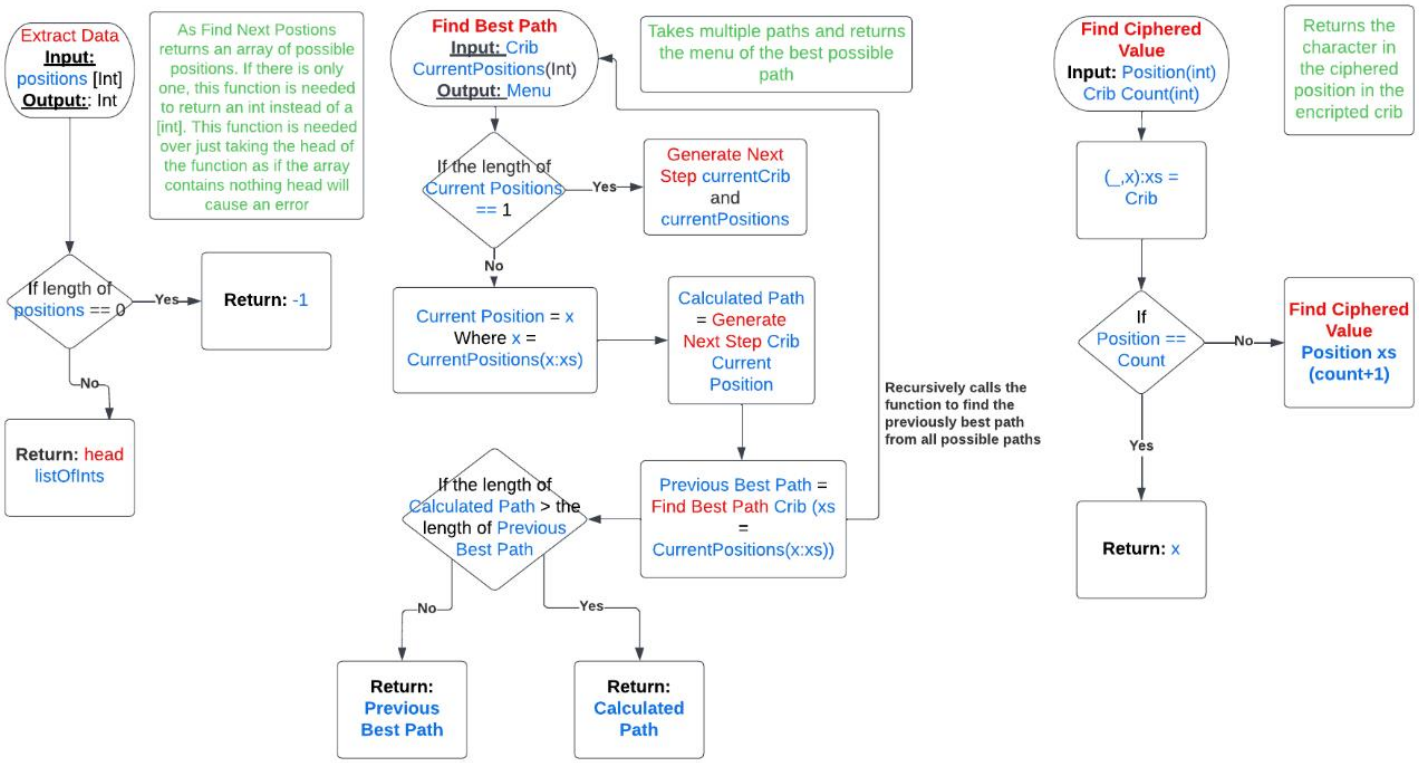

Figure 6 - Longest Menu Design Part 2

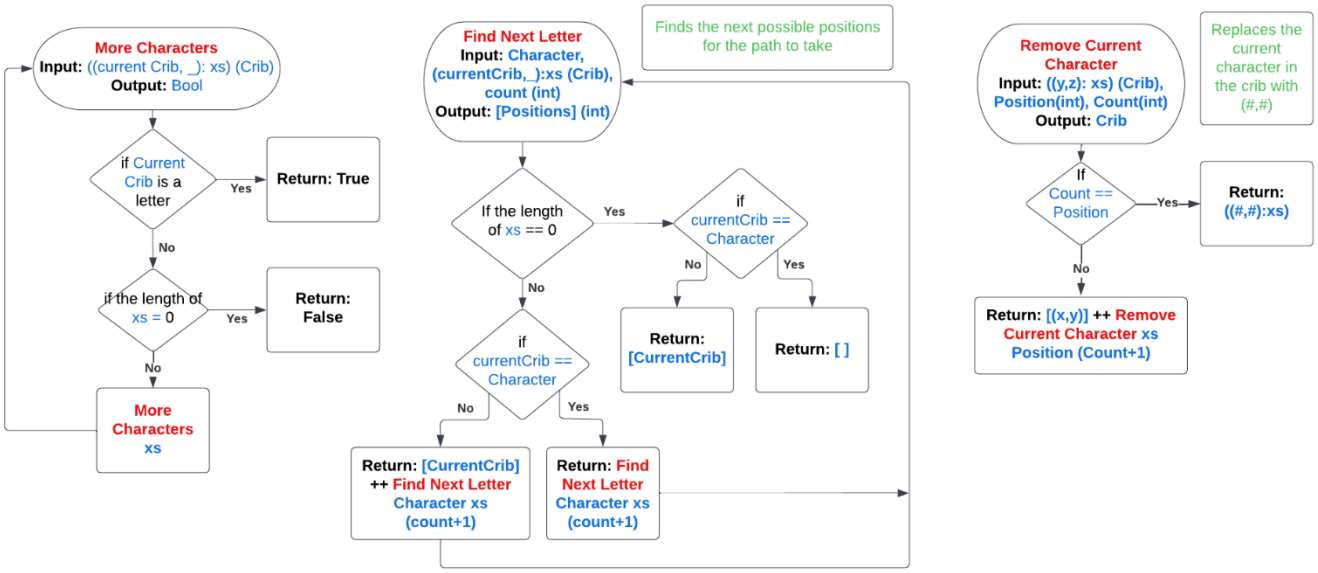

Figure 5 - Longest Menu Design Part 3

Finding The Longest Menu Testing

The first functions that were implemented were the ones that had the least dependencies on other functions. The table below show the order the functions were implemented.

More Characters

Named “moreCharacters” in the code, its purpose was to check if there were any more indexes that had not been used in the current menu. It is able to calculate this as when an index is used it is replaced with (#, #).

Test Input (Crib, Boolean)	Rational Behind Values	Expected	Output
[('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E'), ('O','R'), ('C','D'), ('U','O'), ('M','C'), ('E','U'), ('N','M'), ('T','E')]	Base case check	True	True
[('#','#'), ('#','#'), ('#','#'), ('T','W')]	One remaining value	True	True
[('#','#'), ('#','#'), ('#','#'), ('#','#')]	No possible values	False	False

Find Ciphered Value

Named “findCipheredValue” in the code, this function is quite simple and just returns the equivalent ciphered value for the provided position. It takes in the index to look for, the crib and a counter which is by default 0.

Test Input (Int, Crib, Int)	Rational Behind Values	Expected	Output
0, [('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E')], 0	Base case check	‘A’	‘A’
4, [('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E')], 0	End of range check	‘E’	‘E’
2, [('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E')], 0	Middle Check	‘S’	‘S’

Remove Current Character from Crib

Named “removeCurrentCharacter” in the code. It removes the specified character position from the crib and replaces it with a hash tag. It takes in a crib, the index and a count.

Test Input (Crib, Int, Int)	Rational Behind Values	Expected	Output
[('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E')], 0, 0	Start of range check	[('#','#'), ('E','N'), ('S','S'), ('T','W'), ('D','E')]	[('#','#'), ('E','N'), ('S','S'), ('T','W'), ('D','E')]
[('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E')], 4, 0	End of range check	[('T','A'), ('E','N'), ('S','S'), ('T','W'), ('#','#')],	[('T','A'), ('E','N'), ('S','S'), ('T','W'), ('#','#')],
[('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E')], 2, 0	Middle of range check	[('T','A'), ('E','N'), ('#','#'), ('T','W'), ('D','E')]	[('T','A'), ('E','N'), ('#','#'), ('T','W'), ('D','E')]

Find Next Letters

Named “findNextLetter” in the code, this function identifies the next possible paths that the menu could take based upon the provided letter. It then returns them as an array of ints containing their indexes.

Test Input (Character, Crib, Int)	Rational Behind Values	Expected	Output
‘P’ [('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E'), ('O','R'), ('C','D'), ('U','O'), ('M','C'), ('E','U'), ('N','M'), ('T','E')] 0	Base case of when value is not in crib so no paths are available	[ ]	[ ]
‘S’ [('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E'), ('O','R'), ('C','D'), ('U','O'), ('M','C'), ('E','U'), ('N','M'), ('T','E')] 0	Only one possible path	[2]	[2]
‘E’ [('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E'), ('O','R'), ('C','D'), ('U','O'), ('M','C'), ('E','U'), ('N','M'), ('T','E')] 0	Multiple paths	[1,9]	[1,9]
‘T’ [('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E'), ('O','R'), ('C','D'), ('U','O'), ('M','C'), ('E','U'), ('N','M'), ('T','E')] 0	Three paths	[0,3,11]	[0,3,11]

Find Best Path

Named “findBestPath” in the code, it takes in a crib and the next possible paths to take. It then goes through each possible path and returns the path of the best one.

Test Input (Crib, [Int])	Rational Behind Values	Expected	Output
[('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E'), ('O','R'), ('C','D'), ('U','O'), ('M','C'), ('E','U'), ('N','M'), ('T','E')], [0,3,11]	Test of functionality	N/A	[11, 1, 10, 8, 6, 4, 9, 7, 5]

Extract Int from a list of ints

Named “extractData” in the code. Its purpose is to extract the first element from a list of ints without throwing an exception if the list of ints is empty. In this case it returns -1 so the program knows that the list is empty.

Test Input ([Int])	Rational Behind Values	Expected	Output
[0,1,2,3,4]	Base case	0	0
[ ]	Empty list check	-1	-1

Generate Next Step

Named “generateNextStep” in the code, it takes in an int containing the starting position and then will generate the next steps until it cannot find any further steps, returning a menu at the end. If multiple possible paths are found, it will call findBestPath to resolve this issue. If the provided starting position is -1 the program will return an empty array.

Test Input (Crib, Int)	Rational Behind Values	Expected	Output
[('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E'), ('O','R'), ('C','D'), ('U','O'), ('M','C'), ('E','U'), ('N','M'), ('T','E')], 0	Base case test	[0]	[0]
[('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E'), ('O','R'), ('C','D'), ('U','O'), ('M','C'), ('E','U'), ('N','M'), ('T','E')], (-1)	Return empty list	[ ]	[ ]
[('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E'), ('O','R'), ('C','D'), ('U','O'), ('M','C'), ('E','U'), ('N','M'), ('T','E')], 8	Functionality test	[8,6,4,9,7,5]	[8,6,4,9,7,5]

Find Longest Menu from Provided Crib

Named “findLongestMenu” in the code, its purpose is to iterate through each possible starting position and generate the best path for each index then return the longest path. Its inputs are the crib and the starting index (normally 0).

Test Input (Crib, Boolean)	Rational Behind Values	Expected	Output
[('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E'), ('O','R'), ('C','D'), ('U','O'), ('M','C'), ('E','U'), ('N','M'), ('T','E')], 0	Functionality test	N/A	[11, 1, 10, 8, 6, 4, 9, 7, 5]

Longest Menu

This is the function that is called in main.hs. Its purpose in my implementation is to call “findLongestMenu” and give it a starting index of 0.

Test Input (Crib, Boolean)	Rational Behind Values	Expected	Output
[('T','A'), ('E','N'), ('S','S'), ('T','W'), ('D','E'), ('O','R'), ('C','D'), ('U','O'), ('M','C'), ('E','U'), ('N','M'), ('T','E')]	Same results as findLongestMenu when 0 is provided	[11, 1, 10, 8, 6, 4, 9, 7, 5]	[11, 1, 10, 8, 6, 4, 9, 7, 5]

Critical Reflection

Upon first starting this module, functional programming didn't make sense. I couldn't understand how to use this language to the best of my ability, and the lack of variables posed a significant challenge. Therefore, when it came time to code this project, the planning side of the module became extremely important. Breaking the problem down into smaller, more approachable parts and implementing the least dependent parts first, gave me a way to start the project. Without planning, I wouldn't know where to begin, as I would have run into issues with other modules that had not been implemented yet. This new appreciation of planning has been very impactful on my approach to other modules and how to structure the way I approach coding. A good plan saves more time coding that it costs to create.