Implementierung einer Enigma-Rotorschlüssel-maschine

Lennart Schrader, 70466182

Till Hajek, 70459970

Wie ist die Enigma aufgebaut?

1: Rotierende Walzen

Austauschbar und
in ihrer Stellung veränderbar

2: Anzeige für die Ausgabe

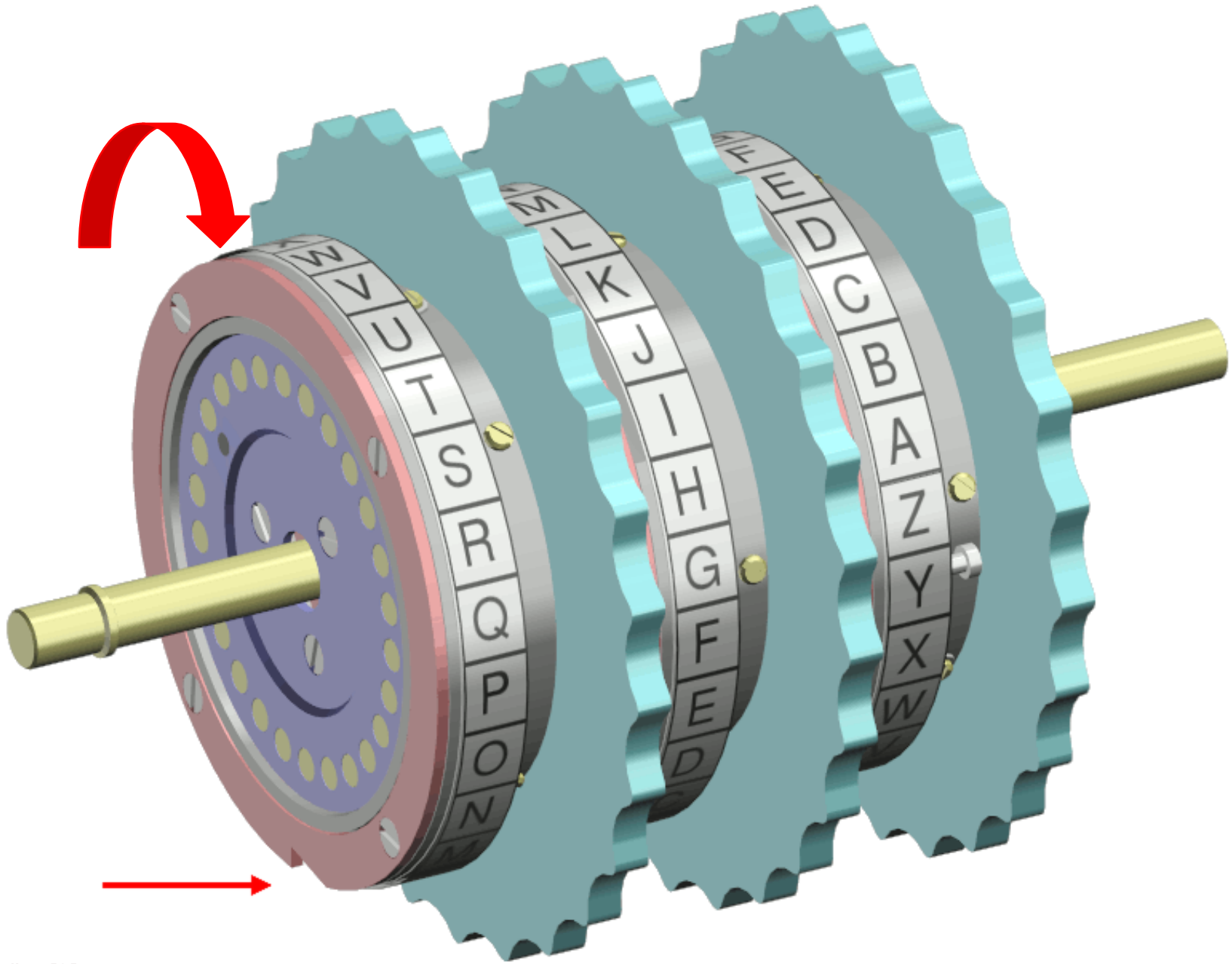
3: Tastatur

26 mögliche Eingaben des lateinis chen Alphabets, keine Zahlen, Sat zzeichen, Leerzeichen

4: Steckbrett/ Plugboard

Umleiten der elektrischen Signale

Quelle: [1]

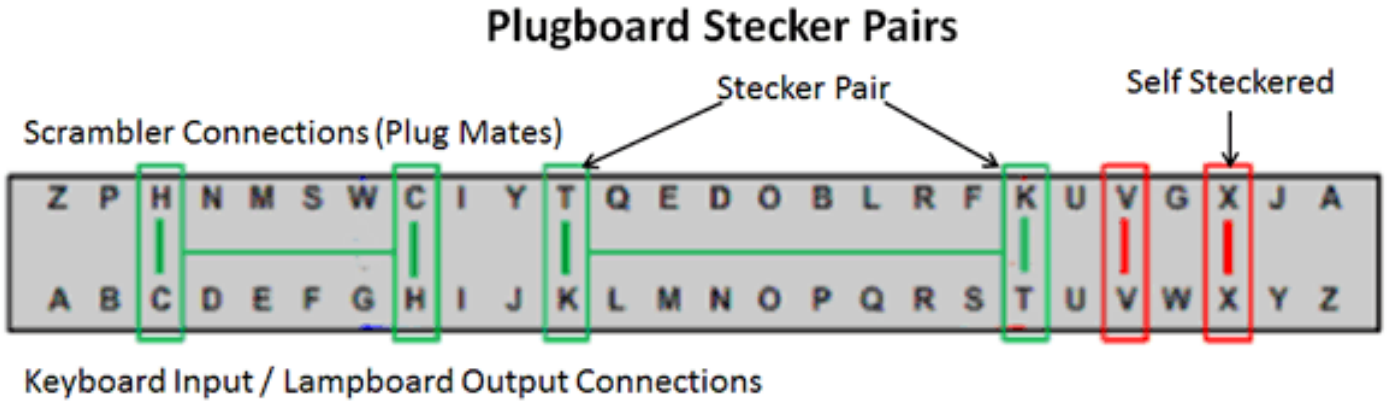


Quelle: [2]

Walzen

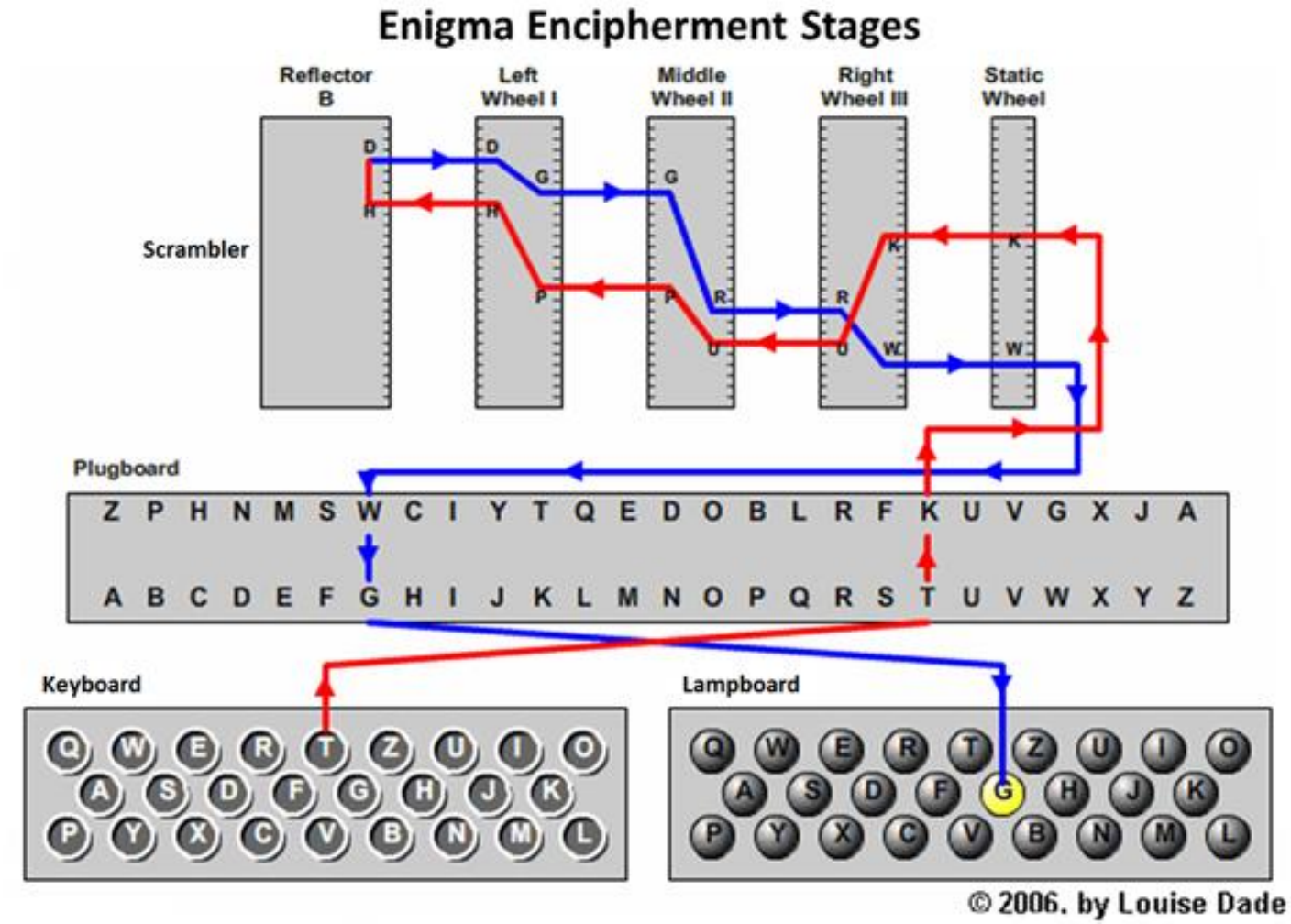
- Kombination der Walzen ist frei wählbar
- "Zufällige" Verdrathung innerhalb der Walzen: Eingabe ≠ Ausgabe
- Erste Walze wird um eine Position nach der Eingabe Verschoben
- 2. und 3. Walze verschiebt sich nach vollständiger Drehung des Vorgängers
- Vollständige Drehung durch Kerbe signalisiert

Plugboard



- Prinzip der Substitution
- Buchstaben werden "getauscht"
- "getauschter" Buchstabe wird weiter gegeben
- Plugboard wird auf dem Rückweg noch einmal durchlaufen

Quelle: [3]



Quelle: [4]

Gesamtsystem

- Zweifaches "Durchlaufen" aller Walzen
- Statische Subsitution durch einen austauschbare Umkehrwalze/ Reflector
- Verschlüsselung erfolgt Zeichen für Zeichen manuell
- Insgesamt 206.651.321.783.174.268.000.000 Verschlüsselungsmöglichkeiten

Implementation in Haskell

```haskell
alphabet :: [Char]
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

-- Data & Types
data Walze = Walze {rand_alphabet:: String, klar_alphabet :: String, umspringbuchstabe ::Char}
    deriving(Show)

type Walzenkombi = (Walze, Walze, Walze)

type Umkehrwalze = [(Char, Char)]

type Plugboard = [(Char,Char)]

-- Walzen
walze1 :: Walze
walze1 = Walze "VZBRGITYUPSDNHLXAWMJQOFECK" alphabet 'K'
```

```haskell
-- Walzenfunktionen
dreheWalze :: Walze -> Walze
dreheWalze (Walze rand_alphabet klar_alphabet umspringbuchstabe) = Walze rand_alphabet_neu klar_alphabet umspringbuchstabe
      where rand_alphabet_neu = tail rand_alphabet ++ [head rand_alphabet]

checkObDrehen :: Walze -> Bool
checkObDrehen walze = umspringbuchstabe walze == head (rand_alphabet walze)

dreheAlleWalzen :: (Walze, Walze, Walze) -> (Walze, Walze, Walze)
dreheAlleWalzen (w1, w2, w3) = (dreheWalze w1,
                                if checkObDrehen w1 == True
                                   then dreheWalze w2
                                       else w2,
                                if checkObDrehen w2 == True
                                   then dreheWalze w3
                                       else w3)
```

components.hs

- Walzen als neues Data Objekt: rand_alphabet steht für das "zufällige" Ausgabe der Walze, klar_alphabet für die Eingabe
- Walzenkombination besteht immer aus drei Walzen in einem 3-Tupel
- Umkehrwalze und Plugboard als Liste aus 2-Tupeln

- Drehen der Walze durh Neuanlegen mittels Konstruktor, Verwendung eines verschobene n rand_alphabet
- Kerbe der Walze wird durch einen festgelegten Buchstaben simuliert
- Ist der Kopf der Liste gleich dem Buchstaben ist die Drehung vollendet

```haskell
sucheInWalze :: [(Char,Char)] -> Char -> Char
sucheInWalze [] _ = '!' -- Fehlerindikator
sucheInWalze tupelliste c | snd (head tupelliste) == c = fst (head tupelliste)
                          | otherwise = sucheInWalze (tail tupelliste) c

uebersetzen :: Char -> Walze -> Char
uebersetzen c walze =  sucheInWalze (zip random klar) c
                    where random = rand_alphabet walze
                          klar = klar_alphabet walze

uebersetzenMitKombi :: Char -> Walzenkombi -> Char
uebersetzenMitKombi c (w1, w2, w3) = uebersetzen (uebersetzen (uebersetzen c w1) w2) w3

zurueckuebersetzen :: Char -> Walze -> Char
zurueckuebersetzen c walze = sucheInWalze (zip klar random) c
                           where random = rand_alphabet walze
                                 klar = klar_alphabet walze

zurueckuebersetzenMitKombi :: Char -> Walzenkombi -> Char
zurueckuebersetzenMitKombi c (w1, w2, w3) = zurueckuebersetzen (zurueckuebersetzen (zurueckuebersetzen c w1) w2) w3
```

```haskell
-- Umkehrwalzen
umkehrwalze1 :: Umkehrwalze
umkehrwalze1 =   zip   "EJMZALYXVBWFCRQUONTSPIKHGD" alphabet

umkehrwalze2 :: Umkehrwalze
umkehrwalze2 =   zip  "IMETCGFRAYSQBZXWLHKDVUPOJN" alphabet

umkehrwalze3 :: Umkehrwalze
umkehrwalze3 = zip  "YRUHQSLDPXNGOKMIEBFZCWVJAT" alphabet

-- Umkehrwalzenfunktion
umkehren :: Char -> Umkehrwalze -> Char
umkehren c umkehrwalze | snd(head umkehrwalze) == c = fst (head umkehrwalze)
                       | otherwise = umkehren c (tail umkehrwalze)
```

components.hs

- Kombinieren beider Alphabet mit der zip Funktion
- Suche mit der Eingabe in klar_alphabet-Teil des Tupels
- Ausgabe des verschlüsselten "Tupel-Partners"
- Dreifache Ausführung des Übersetzen
- Zurückübersetzen des nach der Umkehrwalze erfolgt analog

- Umkehrwalzen werden nicht verändert
- Suche nach der Eingabe in die Umkehrwalze als Schlüssel
- Ausgabe des "Tupel-Partners"

```haskell
-- Plugboard/Streckbrett
plugboard :: Plugboard
plugboard = []

--Plugboardfunktionen
toList :: Char -> Char -> Plugboard
toList c1 c2 = [(c1, c2)]

changePlugboard :: String -> Plugboard -> Plugboard
changePlugboard [] board = []
changePlugboard (x:y:xs) board = toList x y ++ toList y x ++ changePlugboard xs board

initialesVerschluesseln :: Char -> Plugboard -> Char
initialesVerschluesseln c [] = c
initialesVerschluesseln c board | c == fst(head board) = snd (head board)
                                | otherwise = initialesVerschluesseln c (tail board)
```

```haskell
verschluessle :: Char -> Walzenkombi -> Umkehrwalze -> Plugboard-> Char
verschluessle c (w1, w2, w3) u p = initialesVerschluesseln (zurueckuebersetzenMitKombi
                                                            (umkehren
                                                                (uebersetzenMitKombi
                                                                    (initialesVerschluesseln c p)(w1, w2, w3)) u) (w3, w2, w1)) p

verschluessleString :: String -> Walzenkombi -> Umkehrwalze -> Plugboard -> String
verschluessleString [] (w1, w2, w3) u p = []
verschluessleString (x:xs) (w1, w2, w3) u p = (verschluessle x (w1, w2, w3) u p) : (verschluessleString xs (dreheAlleWalzen(w1, w2, w3)) u p)
```

components.hs

- Initiales Plugboard als leere Liste
- Steckverbindungen werden doppelt (Hin- und Rückweg) im Plugboard gespeichert
- Verschlüsselung durch das Plugboard als einfache Subsitution von Zeichen

- Verschlüsselungs-Funktion als Anwendung aller Komponentenfunktionen in richtiger Reihenfolge
- Verschlüsselung einer Eingabe als Anwendung der Verschhlüsselungsfunktion auf jedes Element eines Strings

```haskell
--Funktionen um Eingabestrings in Components zu übersetzen
convertWalze :: Char -> Walze
convertWalze '1' = walze1
convertWalze '2' = walze2
convertWalze '3' = walze3
convertWalze '4' = walze4
convertWalze '5' = walze5
convertWalze  _  = error "Bitte eine andere Walze (1-5) wählen"

convertWalzenKombi :: String -> Walzenkombi
convertWalzenKombi (x:y:xs) = (convertWalze x, convertWalze y, convertWalze (head xs))

convertUmkehrwalze :: String -> Umkehrwalze
convertUmkehrwalze (x:xs) | x == '1' = umkehrwalze1
                          | x == '2' = umkehrwalze2
                          | x == '3' = umkehrwalze3
                          | otherwise = error "Bitte eine andere Umkehrwalze (1-3) wählen"
```

```haskell
--Alle gültigen Eingabekombinationen der Walzen
moeglicheWalzenKombis :: [String]
moeglicheWalzenKombis = [a:b:c:[] |a <-['1'..'5'], b <-['1'..'5'], c <-['1'..'5']]

--Alle gültigen Eingabemöglichkeiten für Umkehrwalzen
moeglicheUmkehrWalzen:: [String]
moeglicheUmkehrWalzen = ["1","2","3"]

--Überprüfen der Eingabe des Plugboards
plugboardCheck :: String -> Bool
plugboardCheck [] = True
plugboardCheck string = even (length string) && nurBuchstaben string && keineDuplikate (sort string) -- lazy evaluation

--Hilfsfunktion für plugboardCheck
keineDuplikate :: String -> Bool -- False wenn es Duplikate gibt
keineDuplikate [] = True
keineDuplikate (x:y:xs) | x == y = False
                        | x /= y && length xs == 0 = True
                        | otherwise = True && keineDuplikate (y:xs)

--Hilfsfunktion für plugboardCheck

nurBuchstaben:: String -> Bool
nurBuchstaben [] = True
nurBuchstaben (x:xs) | x `notElem` ['A'..'Z'] ++ ['a'..'z']= False
                     | otherwise = True && nurBuchstaben xs

--Textfilter welcher um ungewollte Zeichen zu entfernen
filterText :: String -> String
filterText string = [x | x <- string, x `notElem` ",.?!-:;1234567890ß</>_ +*#ÜÄÖüäö@$%&()=\""]
```

interface.hs

- Importieren von des components.hs Module
- Konvertierungsfunktionen für Strings in die jeweiligen Teile der Enigma

- Prüfung gültiger Eingaben durch definierte Listen
- Überprüfung der Plugboard-Eingaben durch Prüfungsmethoden
- Zu verschlüsselnder Text wird bzgl. Ungültiger Zeichen gefiltert
- Überprüfungen sollen die Robustheit des Programms erhöhen

```haskell
-- Main-"Loop"
main :: IO ()
main =
    do putStrLn "Bitte die Walzenstellung (1-5 möglich) angeben | Bsp: >>123"
       walzenString <- getLine
       if  walzenString `notElem` moeglicheWalzenKombis
           then
               do putStrLn "Fehler: Walzenauswahl ungültig!"
                  main
           else
               do putStrLn "Walzenauswahl O.K."
                  putStrLn "Bitte die Umkehrwalze angeben(1-3 möglich) | Bsp: >>1"
                  umkehrwalzenString <- getLine
                  if umkehrwalzenString `notElem` moeglicheUmkehrWalzen
                      then
                          do putStrLn "Fehler: Umkehrwalze ungültig!"
                             main
                      else
                          do putStrLn "Umkehrwalzenauswahl O.K."
                             putStrLn "Bitte die Steckverbindungen (ohne Dopplungen) angeben | Bsp: >>THLS"
                             plugboardString <- getLine
                             if not(plugboardCheck plugboardString)
                                 then
                                     do putStrLn "Fehler: Steckverbindungen ungültig!"
                                        main
                                 else
                                     do putStrLn "Steckverbindungen O.K."
                                        putStrLn "Bitte den zu ver-/entschlüsselnden Text eingeben"
                                        putStrLn "Achtung: Zahlen bitte ausschreiben, Satzzeichen werden entfernt!"
                                        text <- getLine
                                        putStrLn " "
                                        putStrLn "Ergebnis:"
                                        let plugboardVerwendet =  changePlugboard (map toUpper plugboardString) plugboard
                                        putStrLn (verschluesslString (map toUpper (filterText text)) (convertWalzenKombi walzenString)  (convertUmkehrwalze umkehrwalzenString) plugboardVerwendet)
                                        putStrLn "_____ "
                                        putStrLn " "
                                        main
```

interface.hs

- Wechsel zwischen Ein- und Ausgabe auf der Kommandozeile
- Überprüfung der Eingaben
- Rekursiver Aufruf der Main-Funktion bei fehlerhafter Eingabe
- Ausgabe der Ver-/ Entschlüsselung bei korrekter Eingabe aller geforderter Parameter
- Main-Funktion wird abschließend aufgerufen, sodass eine "Schleife" entsteht

Vielen Dank für Ihre Aufmerksameit!

Quellen

- [1] https://upload.wikimedia.org/wikipedia/commons/thumb/f/f8/Enigma_%28209 67055154%29.jpg/800px-Enigma_%2820967055154%29.jpg [Aufgerufen am 31.05.2021]
- [2] https://upload.wikimedia.org/wikipedia/commons/9/9f/Enigma_rotor_set.png [Aufgerufen am 31.05.2021]
- [3] https://www.mpoweruk.com/images/stecker%20pairs.gif [Aufgerufen am 31.05. 2021]
- [4] https://www.mpoweruk.com/images/Enigma%20Diagram.jpg [Aufgerufen am 31. 05.2021]