

# Émuler une machine ENIGMA avec Arduino.

Par Nulentout : Samedi 16 Mars 2024.



Une fois de plus, mon dernier projet ludique vient d'être mis en ligne sur ROBOT MAKER et je me trouve un peu tristounet car une page est tournée. La programmation tournant à l'addiction, surtout les jours d'hiver où il n'y a pas l'appel de l'extérieur, il me fallait trouver un prétexte pour m'engager dans un nouveau projet. Cette fois encore, disposer d'un jouet informatique de plus n'est pas du tout le but de ce tutoriel. *La finalité consiste une fois de plus à cheminer en s'amusant dans le domaine de la programmation d'ARDUINO dans un contexte plaisant* où l'on cherche à coder une application développée pour une carte NANO par l'entremise du langage C++. L'approche sera progressive, et le didacticiel rédigé au fur et à mesure du développement ce qui risque d'engendrer des remises en cause et des retours en arrière. Ce tutoriel ne sera donc pas l'exposé linéaire d'une solution peaufinée et bien au point, mais une promenade logicielle où pas à pas on va tenter de créer un programme cohérent, voir éventuellement comme pour la Machine de Turing une version du pauvre et une version autonome. *J'insiste sur le fait que le petit module électronique décrit n'est absolument pas un but en soi, le bénéfice réside dans le plaisir d'apprendre dans un contexte ludique* qui motive et justifie cette petite réalisation qui reste avant tout un prétexte.

Comme, par exemple, pour la Machine de Turing élémentaire décrite sur ROBOT MAKER que l'on trouve sur <https://www.robot-maker.com/ouvrages/00-amusons-arduino/> avec la version de "luxe" sur <https://www.robot-maker.com/ouvrages/00-machine-de-turing-autonome/> on va avancer pas à pas et progresser par expérimentations sur un certain nombre de petits démonstrateurs. La ligne à suivre reprend la rigueur et la méthode adoptée lors de ce tutoriel. La finalité consiste à respecter des critères qui semblent raisonnables, et ce de façon systématique.

Turing est directement associé à la codeuse ENIGMA, alors chercher à traduire le fonctionnement de cette machine à écrire spéciale sera un hommage de plus à cet illustre mathématicien qui a influencé indirectement et de façon colossale le déroulement du conflit de 1939 à 1945. Aussi,



Fig.1

après avoir réalisé diverses versions de Machines de Turing, le choix de créer une machine ENIGMA virtuelle s'imposait presque naturellement. Il est clair que si vous proposez le mot clef ENIGMA à un moteur de recherche, non seulement vous allez vous retrouver noyé par le nombre des articles qui traitent du sujet. Et surtout vous allez également trouver plusieurs programmes qui simulent cette codeuse, tous très bien faits probablement très supérieurs à ce que je vous propose.

C'est donc uniquement dans l'exercice de programmation d'une "version de plus" que l'on récoltera le fruit de notre investissement. Les simulateurs disponibles sur la toile étant gratuits, il importait que notre investissement soit le plus faible possible. C'est la raison pour laquelle je vous propose une première version qui n'utilise qu'une carte Arduino NANO. C'est le Moniteur de l'IDE qui permettra le dialogue Homme/Machine et il n'y aura dans un premier temps strictement rien à souder. Du reste lorsque vous aurez terminé cette balade informatique, la carte Arduino pourra servir à tout autre chose si vous le désirez.

## 01) Structure d'une machine ENIGMA.

Reproduire en moins bien les innombrables descriptions de cet appareil qui pullulent sur l'Internet ne présente strictement aucun intérêt. On va dans ce premier chapitre résumer le strict minimum pour pouvoir ensuite analyser le fonctionnement de l'appareil électromagnétique et le traduire en langage C++ pour aboutir à un comportement virtuel analogue. Il est hors sujet ici d'aborder l'utilisation opérationnelle de cette codeuse, la richesse d'internet sur ce thème étant considérable. Donc on oublie les diverses version de cette famille de codeuse pour aller au plus simple.

### ➤ Architecture d'une ENIGMA.

Créer un émulateur avec une carte Arduino va consister à agencer des algorithmes qui seront capables de reproduire le comportement de chaque unité fondamentale de cette codeuse, que ce soit électrique ou mécanique. Vous avez largement exploré quelques articles sur internet et vous savez qu'elle ressemble à une machine à écrire et a été conçue pour automatiser le chiffrement par **substitution dynamique**. L'opérateur frappe les caractères du texte à encoder ou décoder sur le **Clavier 4** qui n'offre que les vingt-six lettres de l'alphabet. (*Pas de chiffres, et pas de ponctuation, y compris pas d'espace.*) Outre le clavier, en **3** se trouve le tableau des ampoules électriques de décodage. Lorsque l'opérateur enfonce une touche sur **4**, la lettre commence à transiter à travers les trois **Rotors** de codage **2** du "brouilleur". Chaque rotor transforme la lettre qu'il "reçoit" en une autre lettre. Puis, la lettre sortante du rotor de gauche traverse le **Réflecteur 1** et retourne modifiée pour traverser dans l'autre sens les trois **Rotors** subissant encore trois substitutions. Cette lettre est alors fournie au **TABEAU** de **FICHES 3** où elle est modifiée une dernière fois. Elle est alors présentée au plateau de substitution **qui allume l'ampoule de la lettre codée ou décodée**. Sur la Fig.2 la grille **G** de protection du système est relevée pour permettre à l'opérateur de conditionner la machine en choisissant et en intercalant le **Réflecteur** et les trois **Rotors**. **Il existe plusieurs types d'ENIGMA. Dans ce didacticiel nous allons nous consacrer au modèle M3, celui de la Fig.2 qui était utilisée dans la Kriegsmarine, et en particulier dans les sous-marins U-Boats.** Elle fonctionnait avec trois rotors en prise dans le "brouilleur". Le nombre de rotors disponibles qui pouvaient être insérés dans n'importe quel ordre était de huit. Le logiciel de simulation va devoir émuler chacun de ces modules et permettre à l'opérateur de placer la machine virtuelle dans les conditions initiales permettant le cryptage et le décryptage.

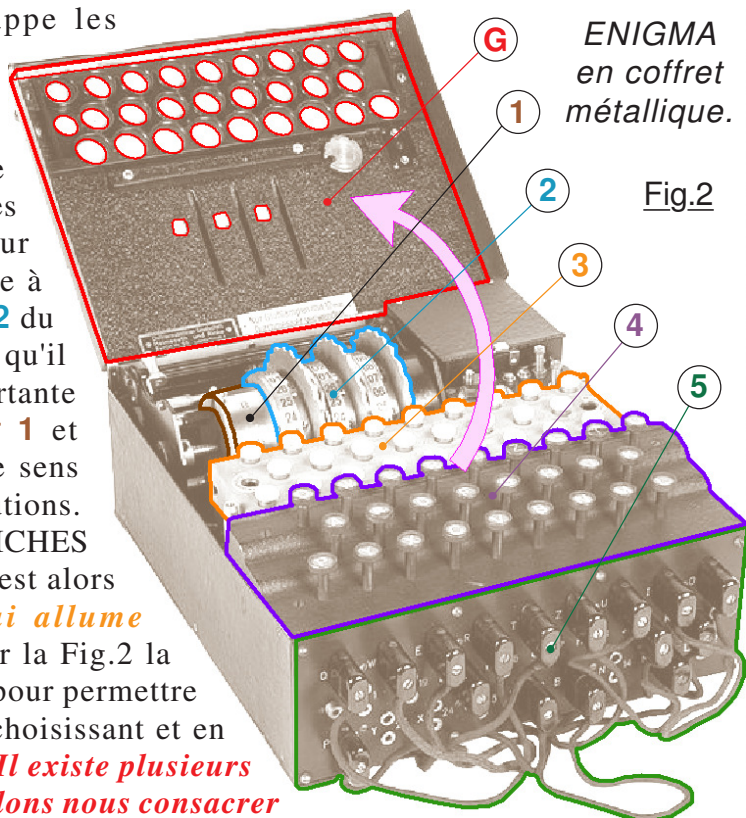


Fig.2

### ➤ Comportement mécanique d'ENIGMA.

**S**ubstitution dynamique des caractères précise qu'une lettre n'est jamais modifiée de la même façon. Si on propose à la machine le texte AAAAAA elle va coder par exemple BWCXLM. Pour assurer cette fonction, chaque fois que l'opérateur enfonce une touche, il agit sur un mécanisme qui décale d'une position la roue codeuse de droite. Quand cette dernière a effectué un tour, elle entraîne à son tour la roue centrale qui s'indexe d'une position voisine. Lorsque cette roue centrale a effectué un tour complet, elle fait à son tour changer de position la roue de droite. Ce mécanisme se comporte donc comme un compteur en base 26. Du coup il faut frapper  $26 \times 26 \times 26$  fois sur le clavier **4** soit 17576 lettres pour qu'un cycle de transposition recommence. L'émulateur que l'on se propose de développer devra simuler ce comportement sachant que la position qui engendre le changement d'index est fixe sur les rotors, mais que le positionnement initial de la machine pour travailler est quelconque et change à chaque jour du conflit à 0H00. Pas mal de cogitations en perspectives ... NANO est-elle assez puissante pour supporter cette "charge" ?

## 02) Fonctionnement électrique d'une machine ENIGMA.

P assage obligé pour décrire le fonctionnement électrique de chaque module, nous allons nous contenter ici du strict minimum. Ce type d'explications pullule sur la toile et si j'en fais ici un résumé ultra condensé, c'est pour vous éviter d'effectuer des recherches et de faire de ce tutoriel un ensemble cohérent et surtout autonome. Ultra produit et reproduit, le schéma de la Fig.4 explicite le cheminement du courant électrique dans la machine et précise la transformation effectuée sur la lettre qui est frappée sur le clavier de l'ensemble. **Le chiffage s'effectue lettre à lettre.** Dans cet exemple, pour le codage initial de la journée envisagée, on a introduit les deux **Fiches croisées** du schéma de la Fig.3 respectivement sur la lettre S et la lettre D. Ainsi, une

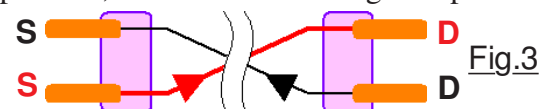


Fig.3

lettre **S** sera transcodée en **D** et réciproquement une lettre **D** sera codée **S**. **On satisfait ainsi le cryptage et le décryptage avec une configuration initiale identique.** Supposons par exemple que l'opérateur radio frappe sur la touche **(A)**. L'électricité partant de **a** et arrivant en **b** transite par le contact travail de l'**inverseur** en **c**. Du plot en **d** sur la prise double **A** le courant traverse la palette **P** poussée sur les deux plots par le ressort symbolisé en vert clair. Partant du plot en **e** le courant se retrouve en **f**. Il traverse alors le **Brouilleur** pour arriver en **g**. (Noter au passage que la ligne noire représente un autre cheminement non utilisé dans cet exemple.) Dans le **Réflecteur** le circuit se poursuit en **h** pour retraverser le **Brouilleur** et ressortir en **i**. La ligne électrique correspondant au barillet de sortie arrive

alors sur la prise double **S**. Hors la **Fiche croisée** pousse avec les broches **b** la plaquette **P** par les isolants **i**. Du coup la tension en **j** transite par la ligne croisée en **k**. Puis de **k** elle arrive alors en **i** sur le commun de l'**inverseur** **D** dont le contact repos alimente l'ampoule **D** en **m**. Lorsque l'opérateur relâche la touche **A** le mécanisme fait changer

le **Rotor** de droite d'une position. **Du coup les croisements internes de la zone colorée en vert pastel changent de configuration.** Si on clique à nouveau sur la lettre **A**, le cheminement ne sera plus identique et la sortie du brouilleur se fera sur une autre ligne mais plus en **i**. Lorsque l'on frappera 26 fois la lettre **A**, chaque fois la transposition dans le **Brouilleur** sera différente. Un caractère **A** de plus et c'est le rotor central qui se décale d'une position. Les 26 lettres **A** qui suivent seront encore différentes. Pour qu'un cycle dans le brouilleur se répète, il faut  $26 \times 26 \times 26 = 17\,576$  caractères frappés sur le clavier. Tant que le message à encrypter ne dépasse pas les 17 576 caractères, il n'y a aucune répétition de l'alphabet de substitution. La Fig.4B précise la répartition des touches sur le clavier ou des ampoules électriques sur le tableau de transcription des caractères. C'est la même clé d'initialisation de la machine qui sert à coder et à décoder. En effet, si la lettre U est transformée en B, réciproquement le B est permuté en U. Dans le cas d'Enigma, cette clé indique le réglage de départ des mécanismes de la machine émettrice. Puisque le codage est réversible, seule la connaissance de la configuration initiale des rotors, du réflecteur et du tableau de connexions de l'Enigma suffit. **Noter que par conception de la machine il est impossible qu'une lettre soit transposée en elle même.**

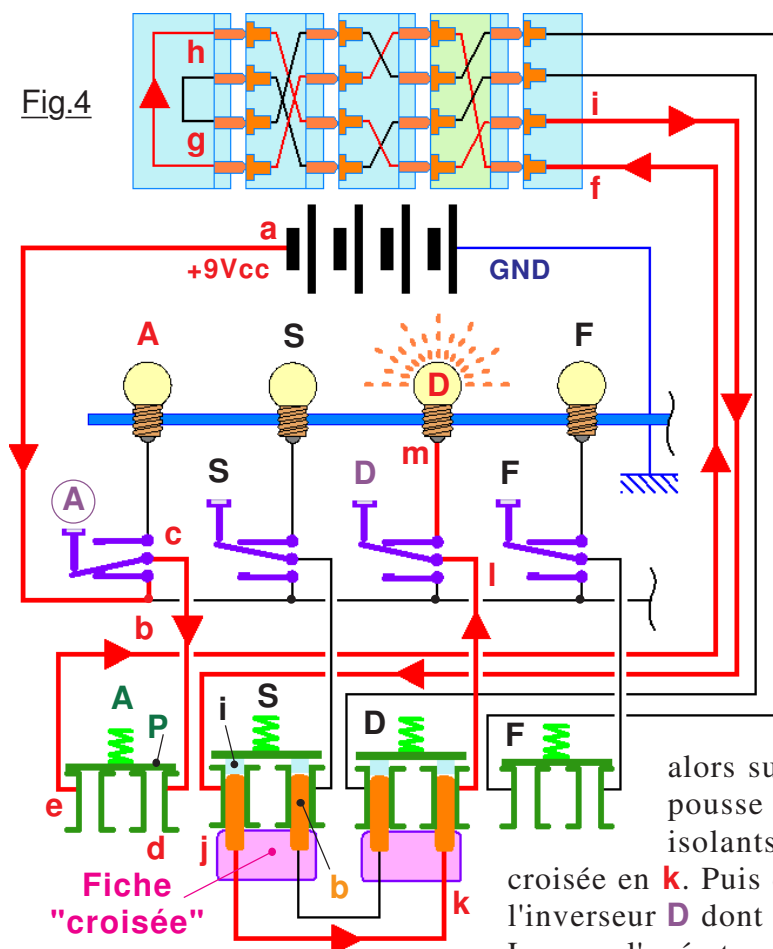
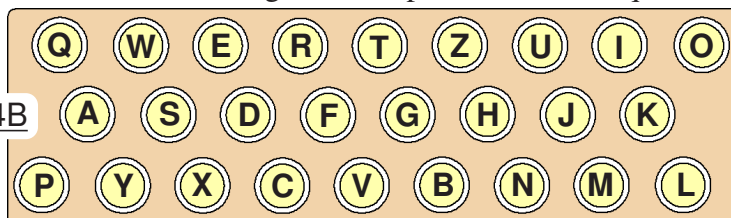


Fig.4

Fig.4B



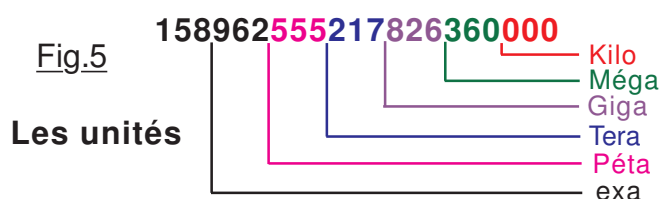


### ➤ La combinatoire explosive d'ENIGMA.

**B**ien que ce ne soit strictement pas utile du tout pour développer un émulateur de cette machine à coder, il me semble important de souligner ici le génie des techniques mises en œuvre pour générer un transcodage dont le nombre de clefs possibles soit tellement grand que l'on puisse affirmer que "craquer" son code par la méthode brute soit totalement impossible. (Et montrer au passage le génie d'Alan Turing qui en 1940 à cassé le code de cette codeuse supposée "inviolable".)

- Le **Brouilleur** permet 17 576 possibilités pour positionner initialement les trois **Rotors**.
- Trois Rotors à utiliser dans un ordre quelconque parmi cinq : 60 possibilités.
- Le tableau de connexions avec les dix câbles permute 20 lettres deux-à-deux, seulement 6 d'entre elles restent inchangées. Cette technique engendre 150738274937250 permutations possibles. Le nombre total de combinaisons, soit le produit des combinaisons des choix des rotors, de leurs positions de départ et du tableau de connexion, est donc de

$17\,576 \times 60 \times 150\,738\,274\,937\,250 = 158962555217826360000$  clefs initiales possibles.



Difficile d'imaginer la grandeur de ce nombre. Et encore, il faudrait le doubler car on dispose de deux **Réflecteurs** différents. Par exemple considérons un **Giga**, soit un milliard. On calculera facilement qu'un milliard de secondes représente 11574 jours. Il nous faut vivre 31,7

années pour compter ces 10000000000 secondes. Et sur la Fig.5 nous n'en sommes qu'aux chiffres violets. Si l'on veut compter l'intégralité du nombre, il faut 159 milliard de fois ces 10000000000 secondes. Le chronomètre en comptant une fois par seconde devrait fonctionner durant 5.039.113.000.379 années. (Il faudra changer ses piles un sacré nombre de fois.) Et pourtant durant le deuxième conflit mondial, Alan Turing a réalisé la prouesse qui dès 1940 cassait le code d'Énigma en moins de deux heures ...

### ➤ La route à suivre pour émuler une ENIGMA.

**D**ans ce tutoriel, le but recherché consiste à nous "amuser" à programmer en C++ d'Arduino tout en cherchant à coder avec méthodes et rigueur. Pour mémoire, on ne veut "rien investir". On va donc se contenter de n'utiliser qu'une carte Arduino NANO. Dans ce contexte, nous n'aurons aucun circuit imprimé à souder sauf si on désire passer à **une version de luxe** et intégrer la petite carte électronique dans un boîtier. (Voir le chapitre suivant.) Toutefois, il faut bien avoir une interface Homme/Machine pour la conditionner et s'en servir au Cryptage / Décryptage. Dans le cadre d'un investissement nul, c'est le **Moniteur de l'IDE** qui va servir à établir le dialogue entre l'opérateur et le module NANO. On peut dans ce cadre établir un plan de bataille. On va devoir :

- 1) Programmer une fonction dialogue par la ligne série USB d'ARDUINO. (C'est cette ligne USB qui alimente la carte NANO, qui permet de téléverser le code et qui par le **Moniteur de l'IDE** va nous autoriser à envoyer des consignes avec le clavier de l'ordinateur et afficher les "résultats".)
- 2) Créer du code qui se comportera comme un **Rotor** de la machine. Il faut envisager cinq Rotors.
- 3) Agencer deux **Réflecteurs** virtuels différents.
- 4) Émuler le tableau de branchement des **Fiches croisée**.
- 5) Organiser le mouvement des rotors lors de la frappe virtuelle des caractères. Il importe de noter que tous ne tournent pas au passage de la même lettre ce qui va compliquer notre tâche :

- Le **Rotor I** provoque la rotation de celui à sa gauche pour la transition de **Q** vers **R**.
- Le **Rotor II** provoque la rotation de celui à sa gauche pour la transition de **E** vers **F**.
- Le **Rotor III** provoque la rotation de celui à sa gauche pour la transition de **V** vers **W**.
- Le **Rotor IV** provoque la rotation de celui à sa gauche pour la transition de **J** vers **K**.
- Le **Rotor V** provoque la rotation de celui à sa gauche pour la transition de **Z** vers **A**.

- 6) **Assembler électriquement** ces éléments virtuels. (Enchaîner les permutations.)
- 7) Afficher la fenêtre des trois **Rotors**.

Le chemin est tout tracé. On peut raisonnablement commencer à programmer en soulignant que mis à part le dialogue sur la ligne série, l'intégralité des séquences de ② à ⑤ seraient directement réutilisables si l'on envisageait un simulateur matériel avec clavier, écran etc.

## Présentation de la carte Arduino NANO.

L'arduino NANO se présente sous la forme d'une minuscule carte qui condense l'intégralité des fonctions d'une Arduino UNO tout en ne mesurant que 1,9 cm x 4,5 cm. La NANO utilise l'ATmega328 en version CMS. Les broches d'utilisation sont séparées pour pouvoir la placer sur une platine d'essais classique. Arduino NANO peut être alimentée soit par le connecteur Mini-USB soit en externe avec +6V à +20V non régulé sur la broche **VIN**. **ATTENTION : Pas de VIN simultanément avec la liaison Mini-USB ou le régulateur 5Vcc local sera détruit.** Alimentée par le connecteur Mini-USB la carte fournit  $\approx 4,9v$  sur la broche **5V** pour alimenter des modules périphériques. Cette broche peut également être alimentée en +5Vcc simultanément avec la prise Mini-USB. On peut ainsi alimenter la carte par la broche **5V**, sur son électronique d'application, tout en branchant en parallèle la ligne USB pour programmer sur site et dialoguer avec le **Moniteur de l'IDE**.



14 broches binaires. (Dont 6 fonctionnant en PWM.)

8 broches d'entrées Analogiques dont 6 pouvant fonctionner en E/S.

Courant maximal par broche de sortie : 40 mA. (Total MAX : 100mA)

L'ATmega328 a 32 Ko pour loger le programme, (Avec 2 KB utilisé pour le bootloader).

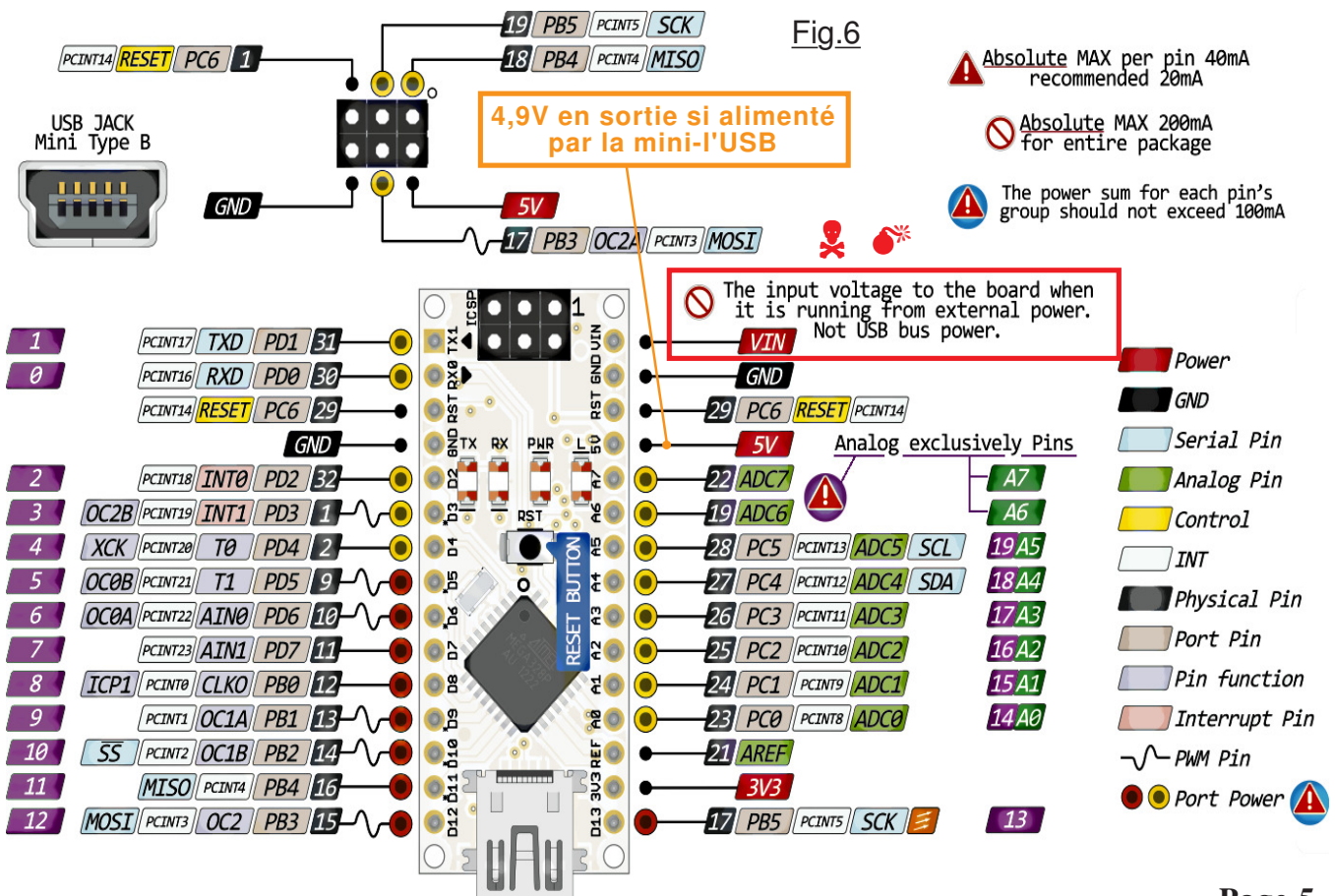
L'ATmega328 a 2 Ko de SRAM pour les variables et 1 Ko de mémoire EEPROM.

Par rapport à la carte Arduino UNO la NANO présente **deux entrées Analogiques supplémentaires A6 et A7**. Elles ne peuvent pas être utilisées en E/S binaires, mais uniquement en entrées analogiques et ne disposent pas de résistances PUL-UP internes. Inutile de les déclarer en entrée, on les utilise directement avec la syntaxe standard `analogRead(20)` et `analogRead(21)`.

Par exemple j'ai commandé un groupe de cinq sur :

[https://www.amazon.fr/gp/product/B078S8BJ8T/ref=ppx\\_yo\\_dt\\_b\\_asin\\_title\\_o01\\_s00?ie=UTF8&psc=1](https://www.amazon.fr/gp/product/B078S8BJ8T/ref=ppx_yo_dt_b_asin_title_o01_s00?ie=UTF8&psc=1)

*Naturellement il n'est pas du tout obligatoire d'en approvisionner cinq d'un coup, mais comme j'en utilise souvent sur de multiples petites applications, j'ai pris de l'avance !*





### 03) Version de luxe de la petite machine ÉNIGMA.

Pour celles et ceux qui désireraient créer un petit objet matérialisant ces études et enfermer la carte Arduino NANO dans un petit boîtier, je vous propose la version de la Fig.7 les fichiers nécessaires pour créer ce boîtier sur une imprimante 3D accompagnant ce descriptif. Du coup, la petite carte électronique doit être placée sur un circuit imprimé décrit plus avant. L'agencement global contribue à minimiser le volume qui sera occupé par l'ensemble de l'électronique. Au final le coffret du petit simulateur montré sur la Fig.7 ne mesure que 62mm de long, 26mm de large et 32mm de haut, feutres situés sur le dessous et bouton de RESET compris. C'est presque la ligne USB qui se branche sur l'ordinateur de dialogue qui prend le plus de place. Les diverses photographies proposées sont trompeuses, en apparence le boîtier semble gros. Dans la pratique ce sont des images saisies en MACRO et le coffret (Voir la Fig.12) est vraiment peu encombrant. Il suffit de savoir que les vis visibles sur

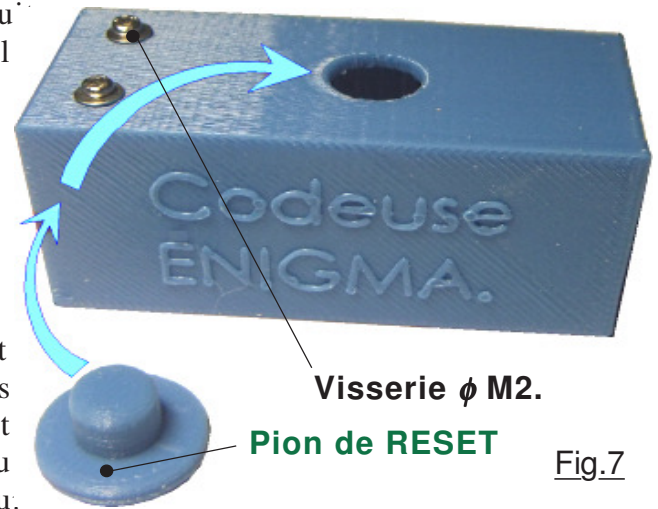


Fig.7

la Fig.7 sont au diamètre nominal de  $\phi$  M2mm, donc très petites. Le schéma électronique retenu et présenté en Fig.8 est "dérisoire" puisqu'il n'ajoute à la carte NANO que deux résistances une LED bleue et un BUZZER actif. Ces deux composants ne sont pas du tout indispensables, vous pouvez les ignorer, mais vu leur faible coût ce serait dommage de s'en passer. C'est juste un petit plus pour

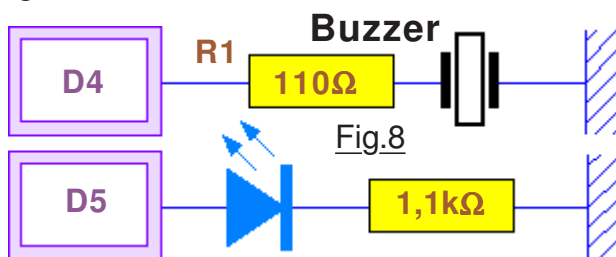


Fig.8

*augmenter les potentialités de du petit ensemble qui pourra servir à tout autre chose.* Vu le peu de composants mis en œuvre il serait possible de se passer de circuit imprimé support. Ceci dit, comme une toute petite carte de prototypage sera suffisante, cette dernière assurera l'immobilisation de l'ensemble électronique dans le coffret. Les sorties D4 et D5 ont

été choisies pour faciliter l'implantation des composants. (*Critère faible.*) Le BUZZER choisi est de type actif pour minimiser le code généré par le compilateur. S'il était directement branché sur D4 les BIPs d'alerte seraient bien trop agressifs, aussi, la résistance R1 diminue le courant envoyé au transducteur lorsque la sortie est à l'état "1". Sur la Fig.9 la carte Arduino NANO n'est pas installée sur les deux lignes de barrette HE14 4 qui servent de support. On peut penser qu'il s'agit d'un luxe couteux. Toutefois, vu que j'approvisionne ces barrettes par lots, leur prix de vente est pratiquement dérisoire. Par ailleurs, la carte Arduino qui a été installée sur ce circuit me sert à développer des

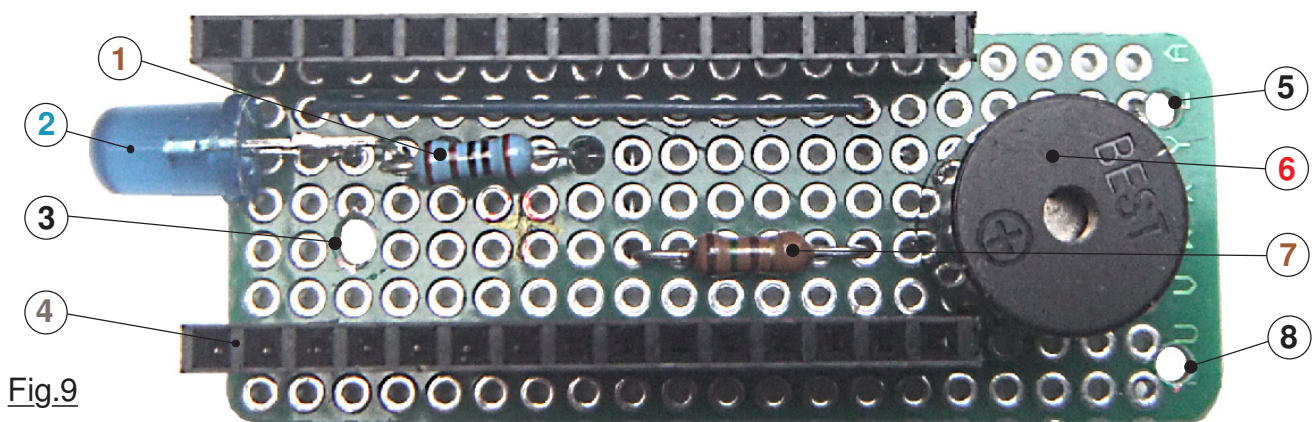


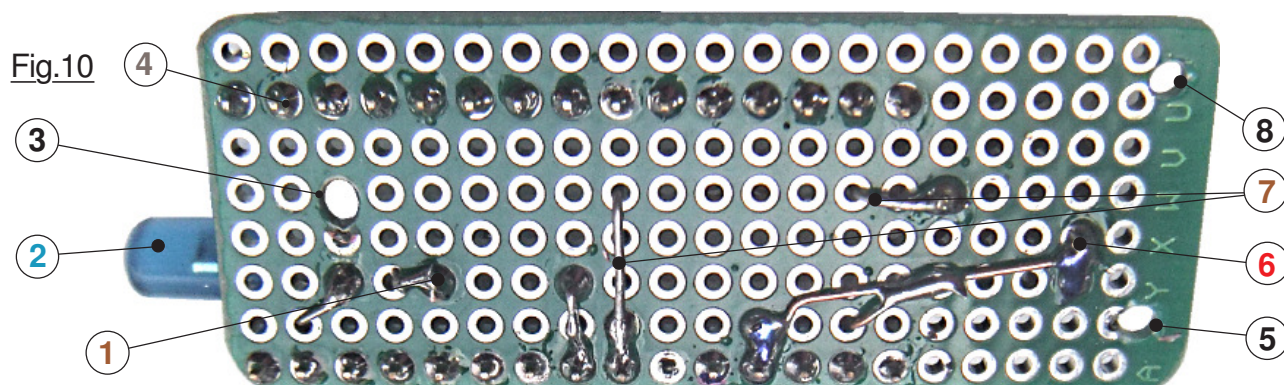
Fig.9

logiciels depuis des années. Aussi, vu le nombre de téléchargement faramineux qu'elle a enduré, il n'est pas exclus que l'on finisse par dépasser le seuil de fiabilité. La remplacer facilement sera rapide, alors que dessouder les 30 broches serait pratiquement infaisable sans y laisser la santé nerveuse. *Par ailleurs, comme la carte Arduino est surélevée, le BUZZER "passe" dessous* ce qui autorise un circuit le plus court possible. On observe la LED bleue en 2 et sa résistance de limitation de courant de 1,1kΩ repérée en 1. En 6 on retrouve le BUZZER actif *polarisé* avec sa

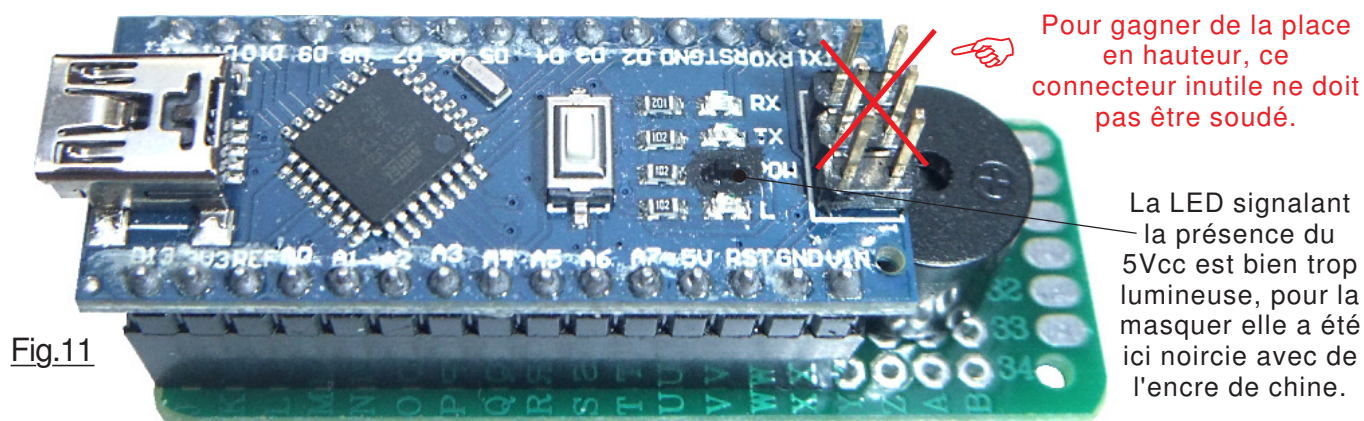
résistance de "limitation sonore" placée en série et repérée par **7**. Le circuit imprimé est supporté en trois points dont les trous de passage des vis  $\phi$  M2 sont bien visibles en **3**, en **5** et en **8**.

### ➤ La vue coté soudures.

**B**anal dans mes réalisations, ce petit circuit imprimé utilise une chute de carte à partilles servant au prototypage, les queues des composants établissant les liaisons électriques. Il me semble inutile de "rabâcher" ici les divers conseils habituels pour réaliser les soudures. La Fig.10 qui reprend certains repères employés sur la Fig.9 est largement suffisante. On soude la résistance **7**, on ajoute le BUZZER **6** puis les deux lignes HE14 telles que celle située en **4**. Pour réaliser la liaison entre **1** et le picot de la ligne HE14 opposé à **4** je me suis contenté de plier la queue de la résistance. On continue par la liaison **7** qui dans la pratique est constituée de la queue de **R1** pliées au ras des pastilles de la sérigraphie et coudée à angle droit pour aboutir au picot du BUZZER. Enfin on complète par la liaison du BUZZER vers GND constituée d'un petit fil de cuivre entièrement dénudé.



On ajoute la LED **2** et sa résistance en **1**, la liaison vers GND étant réalisée par le petit fil bleu isolé pour aboutir au résultat visible sur [Image 01.JPG](#) préservée dans le dossier <Galerie d'images>. Noter que la valeur de **1,1k $\Omega$**  adoptée sur le prototype est déterminée expérimentalement pour obtenir une luminosité correcte tout en consommant le courant le plus faible possible. On complète nos observations en visualisant [Image 02.JPG](#). Sur [Image 03.JPG](#) on a inséré sur les rampes HE14 **4** une carte NANO sur laquelle le connecteur de dessus n'a pas été soudé, ce qui est plus simple. Surtout, on peut y remarquer dans le médaillon rouge *la liaison* qui alimente la LED "**POW**" trop présente *qui a été coupée*. En effet, ce témoin de la présence du **+Vcc** est inutile. En revanche, il parasite fortement l'observation de l'éclairement de la LED en **D13** et surtout des deux témoins de **TX** et de **RX**. Comme montré sur la Fig.11 si on avait déjà soudé le connecteur du dessus, il faut couper ses broches, et *gagner* ainsi du volume *en hauteur indispensable pour que l'ensemble passe*



*dans le petit coffret*. Sur [Image 04.JPG](#) on observe que vers l'arrière les orifices de passage sont assez grand pour serrer l'écrou  $\phi$  M2 avec un outil cloche ce qui facilite grandement l'assemblage final. En particulier sur [Image 06.JPG](#) est soulignée la présence d'une entretoise vers l'avant. Pour l'assemblage, la tête de vis dépasse sur le dessous ce qui oblige à coller les plaquettes de feutre adhésif. La photographie d'[Image 05.JPG](#) représente les deux vis de liaison arrière relativement longues. ( $L = 25\text{mm}$  sous tête.) [Image 07.JPG](#) à [Image 09.JPG](#) complètent les vues du prototype. On applique **+5Vcc** entre la lyre pour la broche de **D4** et GND, le BUZZER doit couiner. On injecte ensuite le **+5Vcc** sur la lyre de la broche de **D5** et la LED bleue doit s'illuminer.

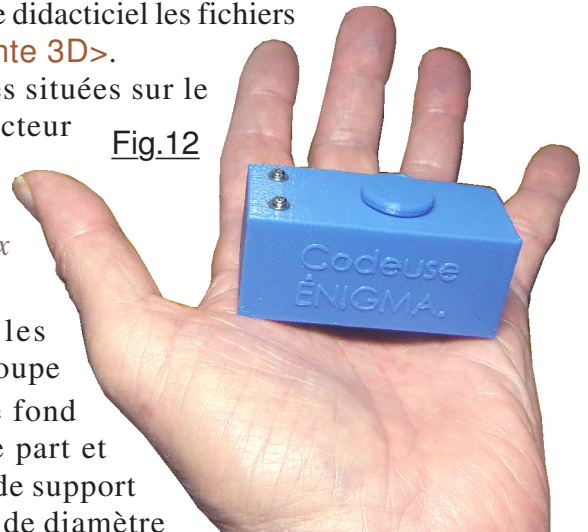


#### 04) Réalisation d'un petit coffret.

P hase ultime de ce projet, je n'ai pas résisté au plaisir de mouler sur mon imprimante 3D un petit boîtier qui englobe entièrement le module électronique, ne laissant visible que la prise USB qui sera reliée par le cordon adaptateur à celle de l'ordinateur. Franchement, on pourrait facilement s'en passer, mais en ce qui me concerne c'est la cerise sur le gâteau. Un simple petit regard sur la Fig.12 prouve que la concrétisation matérielle de ce simulateur d'ENIGMA est bien modeste et n'encombrera pas du tout le plan de travail. Accompagnant ce didacticiel les fichiers relatif au moulage du coffret sont dans le dossier **<Imprimante 3D>**.

**ATTENTION :** Pour gagner un peu en hauteur, les broches situées sur le dessus de la carte NANO ont été coupées au ras du connecteur HE14, car elles ne servent pas. Cette précision était déjà présente sur la Fig.11 donné en page précédente. (*Donc ne pas les souder lorsque vous assemblerez les deux connecteurs HE14 mâle sur votre exemplaire.*)

Fig.12



S ans respecter avec une rigueur absolue toutes les dimensions du coffret, la Fig.13A le présente en coupe pour préciser un peu mieux la solution moulée en 3D. Le fond du boîtier est muni de deux bossages latéraux **1** situés de part et d'autre du BUZZER situé en **2**. Ces deux bossages servent de support au circuit imprimé et sont traversés par les deux boulons **3** de diamètre  $\phi$  M2. Sur le dessous on trouve deux trous **4** facilitant l'introduction des rondelles et des écrous, et surtout de la tête d'une clef de serrage en cloche. En **5** le petit boulon  $\phi$  M2 traverse le dessous et immobilise le circuit imprimé qui s'appuie sur l'entretoise grise **11**. La petite électronique est ainsi solidement bridée à l'intérieur du coffret dont la partie supérieure présente en **6** un orifice de diamètre 10mm pour laisser passer un quelconque stylet prévu pour agir sur le bouton de RESET.

P ersonnellement je trouve peu commode d'introduire un tel dispositif et d'avoir à doser l'effort pour ne pas exagérer les contraintes sur le micro-switch. Aussi, également moulé en 3D le petit bouton de RESET **7** a été calculé de telle façon qu'en appuyant dessus jusqu'à ce qu'il porte sur le dessus du coffret, il déclenche le RESET avec certitude, sans pour autant amener le mini-bouton poussoir en butée mécanique. On retrouve en **8** les six broches du dessus qui ont été coupées le plus court possible sur le connecteur HE14. La tête de vis **5** dépasse sur le dessous rendant le coffret instable sur sa base. Aussi, deux morceaux de feutre autocollants sont ajoutés en **9** et en **10**. Celui situé en **9** est placé le plus à l'arrière possible sans pour autant masquer les trous de passage **4**. Celui placé en **10** est percé d'un trou central pour dégager la zone de la vis et de la rondelle **5**. Dans le dossier **<Galerie d'images>** plusieurs photographies complètent cette description.

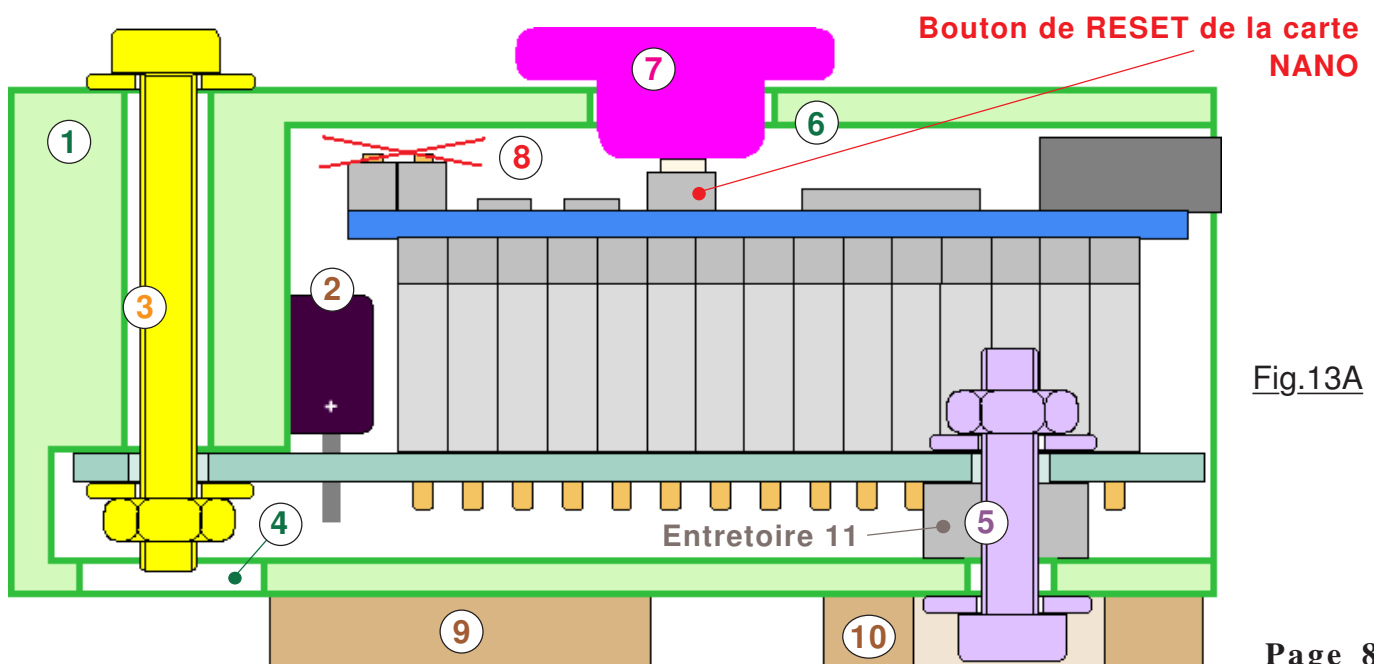
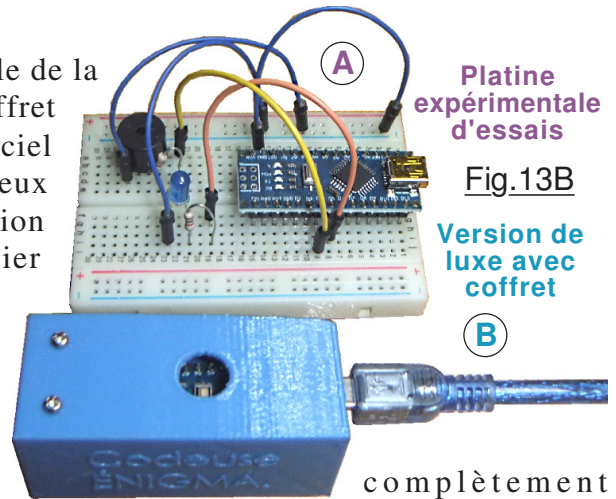


Fig.13A



## 05) Vérification matérielle.

**Q**ue vous soyez tentés par la solution expérimentale de la Fig.13B en **A** ou pour la version de luxe avec coffret en **B** dans les deux cas les programmes de ce didacticiel seront strictement identiques. Que ce soit les nombreux démonstrateurs de ce tutoriel ou le logiciel d'exploitation définitifs, tous ces "sketch" sont logés dans le dossier nommé **<Les programmes Arduino>**. Avant de commencer à aligner du code C++ en vue d'émuler les organes électromécaniques d'ÉNIGMA, il est prudent de vérifier l'intégralité du bon fonctionnement du matériel. **On devrait toujours commencer un projet par définir avec précision sa structure matérielle, agencer cette dernière et surtout en vérifier informatiquement son bon fonctionnement.** C'est indispensable pour pouvoir ensuite programmer en étant certain que si le comportement attendu n'est pas celui observé, c'est bien le logiciel qui est en cause et non le matériel. C'est précisément le but du démonstrateur **P01\_Tester\_le\_materiel.ino** prévu pour valider le bruiteur et la LED bleue.



### ➤ **Anticiper une version de luxe.**

**S**ignalé en page 4, on peut désirer ne rien investir et se contenter uniquement d'une carte Arduino NANO reliée au P.C. par sa ligne USB de téléversement. Dans ce cas ce petit module pourra servir à une autre application lorsque vous aurez exploré les exercices de ce didacticiel. Toutefois, si pour une somme relativement dérisoire vous optez pour la présence du bruiteur, on va pouvoir s'amuser à générer en code Morse l'équivalent de la lettre qui sera transmise au logiciel à partir du **Moniteur de l'IDE**. (*Lettre en Morse une fois qu'elle aura été cryptée ou décryptée par la codeuse.*) Partant de l'idée que ce petit amusement sera possible pour la version définitive du logiciel d'exploitation, autant dès ce démonstrateur créer les routines de traduction en code Morse.

### ➤ **Programmer avec méthode.**

**B**ien que nous soyons dans un domaine ludique, le but fondamental de ce cheminement consiste à explorer ensemble des pistes de codage en langage C++ ayant pour cible les cartes du monde Arduino ; ce n'est plus le résultat qui compte, mais un cheminement informatique qui se veut rigoureux et optimal. Autrement dit, on va chercher à minimiser à outrance le code et programmer **"avec méthode"**. Dans ce cadre particulier, on doit impérativement commencer par choisir des identificateurs "évidents", **la première qualité d'un logiciel quel qu'il soit est sa LISIBILITÉ**. Envisager que votre "production" doit être compréhensible par "tous", et surtout par vous si vous devez reprendre l'ouvrage quelques semaines ou quelques années plus tard. **Un sketch comportera inévitablement des calculs non évidents, ou des astuces d'optimisation. Dans ce cas il sera primordial de les expliciter avec des commentaires.** Éviter autant que possible les séquences très longues imposant des défilements du listage sur l'écran de l'ordinateur. **Idéalement, une procédure ou une fonction ne devrait "jamais" dépasser en longueur la possibilité d'affichage de l'écran pour avoir toutes les instructions simultanément visibles.** Pour illustrer ces conseils nous allons commencer dès le démonstrateur **P01\_Tester\_le\_materiel.ino** pour lequel en outre on va chercher également à optimiser le choix du type des variables utilisées.

**B**ien que ce ne soit pas une obligation du tout, **il est fortement recommandé de placer toute fonction ou procédure avant celle qui y fait appel.** Ainsi, en déverminage, on sait que si une séquence est invoquée, il faut la rechercher en amont. Cette recommandation qui est une obligation en langage PASCAL par exemple fait gagner un temps fou quand on détermine ou quand on améliore un programme "long" c'est à dire incluant des centaines d'instructions.

**Pour structurer "proprement" les constantes et les variables utilisées par le logiciel, elles sont classées par type et globalement par ordre alphabétique. Elles sont de plus énoncées par tailles croissantes : byte, int, long, float, tableaux ... Toutes les déclarations sont placées en tête de programme.**

### ➤ Stratégie adoptée pour la programmation.

Comme avec ce démonstrateur **P01\_Tester\_le\_materiel.ino** on va anticiper la fonction qui génère du Morse dans le petit bruiteur passif, il importe dès cette phase d'adopter une technique homogène avec ce qui sera le programme d'exploitation futur. Après une analyse initiale, je suis arrivé à la conclusion qu'une lettre traitée sera codée par un nombre conformément au tableau de la Fig.14 où *les lettres sont codées directement par leur place dans l'ordre alphabétique*. Je vous laisse le soin de décoder 2/15/14/10/15/21/18. Cette technique de transposition directe des lettres par des nombre présente divers avantages. Que ce soit un **char** ou un **byte** dans les deux cas la place occupée en mémoire sera la même. En outre, pour agencer les tableaux de codage des **Rotors** et du **Réflexeur** on utilisera directement le **byte** du code de la lettre comme indice de sélection des positions dans les variables tableaux qui représenteront ces éléments de la machine de codage.

A	B	C	D			W	X	Y	Z
1	2	3	4			23	24	25	26

Fig.14

Pour traduire les lettres en Morse on va pour chacune utiliser un **byte** comme précisé sur la Fig.15 dans lequel le nombre binaire en violet **N** indique le nombre d'éléments dans le caractère. Les BITS à "0" représenteront des points, alors que les BITS à "1" traduiront des traits. Ces éléments seront lus de la droite vers la gauche. Par exemple sur la Fig.15 on a codé la lettre **B**. Toutes les

Fig.15

0	1	0	N.U.	0	0	1	0
---	---	---	------	---	---	---	---

lettres dans l'alphabet Morse ne compte au maximum que quatre éléments. De ce fait trois BITS sont suffisants pour

indiquer le *nombre d'éléments du caractère* codé. Quatre BITS seront utilisés pour traduire en binaire les points et les traits. Un tableau ordonné de 26 éléments de type **byte** sera suffisant pour transcoder l'intégralité des lettres. Le démonstrateur étant téléversé sur notre simulateur, Le mot proposé en exemple est entendu dans le bruiteur, puis la LED en **D13** d'Arduino et la LED bleue ajoutée vont clignoter à une cadence rapide. Le matériel étant validé, on va pouvoir commencer à construire informatiquement notre machine ÉNIGMA.

Avant de poursuivre, il me semble important de justifier l'utilisation d'un BUZZER de type actif. Le composant d'apparence strictement équivalente montré sur la Fig.16 existe aussi bien en passif qu'en actif. Le type passif a été éliminé pour les raisons suivantes :

- Il impose de générer un signal périodique ce qui impose un code objet plus important, surtout si l'on utilise l'instruction du genre **tone(BUZZER,3000)**. Avec cette instruction *la taille du programme augmente de 1312 Octets d'un coup et 23 Octets de plus dans les données dynamiques*. On pourrait utiliser la **PWM** mais les deux fréquences disponibles sont relativement basses.
- Le composant est un solénoïde qui sous 5V présente une impédance de 11Ω ! Du coup il se comporte presque un court-circuit sur la sortie binaire et à ce régime l'ATmega328 ne résisterait pas très longtemps. Il faudrait intercaler un petit transistor amplificateur de courant ...

Conclusion : Nous allons employer le composant passif de la Fig.16 dont la consommation reste très inférieure à ce que peut fournir Arduino. Sa Tonalité d'environ 3000Hz est idéale et l'activer n'imposera que peu de code objet. À toute fin utile, je vous livre l'adresse où j'ai approvisionné ce composant. (Vous constaterez qu'il est difficile de trouver des liens où on ne livre qu'un seul composant ... et c'est souvent plus coûteux.) :

[https://www.amazon.fr/gp/product/B00GX6YCB1/ref=ppx\\_yo\\_dt\\_b\\_asin\\_title\\_o02\\_s00?ie=UTF8&psc=1](https://www.amazon.fr/gp/product/B00GX6YCB1/ref=ppx_yo_dt_b_asin_title_o02_s00?ie=UTF8&psc=1)

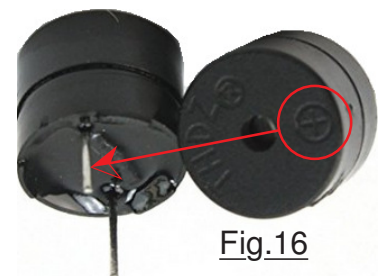


Fig.16

### ➤ Fonctionnement et utilisation du démonstrateur P01.

Étant donné que le but de ces manipulation réside dans l'exercice de la programmation structurée en Arduino, il me semble incontournable de détailler certaines séquences du démonstrateur **P01\_Tester\_le\_materiel.ino** que vous téléversez sur votre carte NANO. Soulignons au passage que *l'intégralité des démonstrateurs ainsi que le programme d'exploitation sont totalement compatibles avec une carte de type UNO* car ils n'utilisent pas les deux entrées spécifiques. Si la NANO est sélectionnée dans ce projet, c'est à la fois pour des raisons financières (*Un module NANO est bien moins couteux qu'une carte UNO.*) et surtout pour sa compacité.



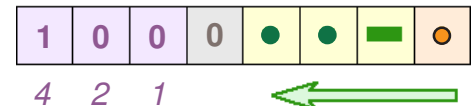
Ordre	Lettre	BINAIRE	Décimal	Morse
1	A	01000010	66	.-
2	B	10000001	129	-.--
3	C	10000101	133	-.-.
4	D	01100001	97	-. .
5	E	00100000	32	.
6	F	10000100	132	..-. .
7	G	01100011	99	-. .-
8	H	10000000	128	....
9	I	01000000	64	..
10	J	10001110	142	.-.-
11	K	01100101	101	-. -.
12	L	10000010	130	.-..
13	M	01000011	67	-. -
14	N	01000001	65	-. .
15	O	01100111	103	---
16	P	10000110	134	.-.-.
17	Q	10001011	139	-.-.-
18	R	01100010	98	.- .
19	S	01100000	96	... .
20	T	00100001	33	- .
21	U	01100100	100	..- .
22	V	10001000	136	...-
23	W	01100110	102	-. -.
24	X	10001001	137	-. -.
25	Y	10001101	141	-. -.-
26	Z	10000011	131	---.

Compte tenu de la structuration des données proposée en Fig.14 et en Fig.15 pour transposer un caractère alphabétique en code Morse, le tableau donné ci-contre résume les diverses représentations de ces entités. C'est le BIT de droite qui sera l'élément convertit en **TRAIT** ou en **POINT** pour la transposition sonore. Du coup le codage des éléments se fait en sens symétrique comme montré sur la Fig.17 sur laquelle sont ajoutés *les poids binaires* des divers BITS qui indiquent le nombre d'éléments dans le caractère.

*BIT lu pour transposer en point ou en trait*

Code Morse pour L = ● — ● ●

Fig.17



Codage en sens réciproque 0 0 1 0

Pour extraire le nombre d'éléments du caractère de sa représentation binaire on utilise les opérateurs et les masques logiques.

### ► Le domaine MASQUES LOGIQUES.

**F**réquentant depuis de longues années des programmeurs amateurs, force est de constater que les si les opérateurs booléens tel que le OU et le ET sont bien assimilés, il en est tout autrement pour l'usage des masques

logiques. C'est particulièrement vrai quand on programme en langage évolué comme BASIC, PASCAL ou C++ etc. Généralement des expressions comme :

- Si (Condition 1) **OU** (Condition 2 ) faire Action 1 Sinon faire Action 2. } sont bien
- Si (Condition 1) **ET** (Condition 2 ) faire Action 1 Sinon faire Action 2. } assimilées.

Il en va tout autrement de l'usage du **OU Exclusif**. Par exemple que signifie :

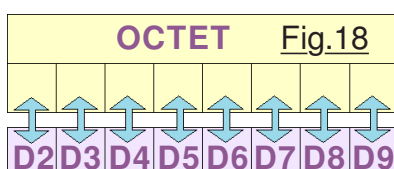
- Si (Condition 1) **OU Exclusif** (Condition 2 ) faire Action 1 Sinon faire Action 2 ? ? ?

C'est la pratique de la programmation "au raz de la machine" notamment en ASSEMBLEUR qui généralise l'usage des **opérateurs LOGIQUES** BIT à BIT et tout particulièrement l'intervention des **masques logiques**. Pourtant, comme on va le voir dans l'exemple de **P01** ce type de programmation peut rendre de signalés services également dans les langages évolués. (*On oubliera ici les opérateurs de décalage utilisés plusieurs fois dans ce petit projet s'ils sont utilisés pour multiplier ou diviser rapidement par des multiples de deux.*) Nous allons dans ce chapitre passer en revue le domaine d'application des masques logiques et des opérateurs associés. On peut déjà noter que :

Les Masques et les opérateurs LOGIQUES servent à **agir individuellement sur des BITS** dans des données globales telles que les **byte** et les **int**, les **long** etc.

### ► La modification d'un OCTET BIT à BIT.

**B**ien que ce chapitre prend en exemple des données de huit BITS, on peut généraliser sans autre forme de procès à des variables entières de taille quelconque. Par exemple on va raisonner sur



une variable **OCTET** qui dans une procédure adaptée sert à lire ou écrire directement les huit BITS des broches binaires allant de **D2** à **D9** comme montré sur la Fig.18 donnée ci-contre. L'instruction suivante **OCTET = DATA** recopiera dans les huit BITS de sortie d'**OCTET** ceux de la variable **DATA**. Réciproquement,

The diagram illustrates the construction of a 16-bit logical mask (Masque\_LOGIQUE) from two 8-bit masks (Masque\_8bits) using logical operations. The diagram shows three examples: ET (AND), OU (OR), and OU Exclusif (XOR). Each example shows the input masks, the operation, and the resulting 16-bit mask.

**Example 1: ET (AND)**

Input 1 (Masque\_8bits): 0 1 0 1 0 1 0 0

Input 2 (Masque\_8bits): 0 1 0 1 0 1 0 0

Operation: ET (AND)

Result (Masque\_LOGIQUE): 0 1 0 1 0 1 0 0

**Example 2: OU (OR)**

Input 1 (Masque\_8bits): 1 1 1 0 0 0 1 1

Input 2 (Masque\_8bits): 1 1 1 0 0 0 1 1

Operation: OU (OR)

Result (Masque\_LOGIQUE): 1 1 1 0 0 0 1 1

**Example 3: OU Exclusif (XOR)**

Input 1 (Masque\_8bits): 1 1 1 0 0 0 1 1

Input 2 (Masque\_8bits): 1 0 1 1 0 1 1 1

Operation: OU Exclusif (XOR)

Result (Masque\_LOGIQUE): 1 1 1 0 0 1 1 1

Sur la Fig.19 la ligne **A** correspond à un **Masque LOGIQUE** commun affecté aux trois exemples. Sur la ligne **B** trois fois la même données sur laquelle est appliqué l'opérateur. En **C** est affiché le résultat de l'opération. Dans l'exemple avec le **ET**, supposons que les huit sorties BINAIRES de **OCTET** pilotent des LEDs. Et bien cette instruction va éteindre **D2**, **D4**, **D8** et **D9** sans modifier les autres sorties. Si on exécute une deuxième fois l'opération sur cette donnée **OCTET** modifiée en **C**, avec le même **Masque LOGIQUE** on constate sur la ligne **D** que l'on ne retrouve pas l'état initial du BIT concerné. La donnée initiale est définitivement perdue car il est impossible d'en retrouver la

- ☞ Ces diverses opérations se font BIT à BIT sur la totalité de la donnée modifiée.

**V**ous avez certainement compris que l'opérateur **OU** qui force des "1" individuellement va servir à piloter des sorties binaires individuellement. Si on désire mettre en marche le moteur (*Ou la LED etc.*) branché sur **D7** il suffit d'un **OU** avec **00000100**. Pour le stopper c'est **ET** associé à **11111011** qui servira de masque. Il nous reste encore à passer en revue le domaine d'exploitation classique du **OU Exclusif**. On a vu en Fig.20, ce que confirme l'exemple de droite de la Fig.19 que cet opérateur sert à inverser les BITS correspondant lorsque dans le **Masque LOGIQUE** on a placé des "1". Du coup on peut inverser l'état d'un système sans se préoccuper de son état initial. C'est exactement ce que font sans que nous le sachions des instructions de type :

La première instruction par exemple inverse l'état de la LED de **D13** une fois par seconde. Dans ces deux instructions, l'opérateur d'inversion d'état qui en C++ est symbolisé par "!" génère en interne pour le microcontrôleur une opération élémentaire avec un OU Exclusif et un masque logique.

Il me semble important de vous faire remarquer que la ligne relative au clignotement de la LED d'Arduino ne fonctionne correctement que par le fait qu'un OU Exclusif appliqué deux fois sur la même donnée avec un Masque LOGIQUE identique restitue cette dernière. Du coup on obtient bien une alternance d'état sur une paire d'exécution puis le cycle se poursuit.

**Page 12**



## > Utilisation classique de l'opérateur ET.

Avant de passer à la suite, examinons ensemble une application banale en informatique de l'opérateur **ET** pour *transformer un texte saisi en minuscules par des majuscules*. Par exemple nous allons utiliser cette technique pour le dialogue Homme/Machine abordé dans le prochain chapitre. Chaque commande est obtenue par un caractère et il est précisé dans l'encadré de la Page 17 que *Quel que soit la touche frappée on ne sera pas obligé de passer en MAJuscule*. Par exemple l'utilisateur frappe 'E' pour dans le protocole adopté corriger une donnée. Vous désirez que cet usager ne soit pas forcément obligé d'enfoncer la touche [MAJ]. S'il frappe 'e', le programme devra le transformer en 'E'. En codage ASCII c'est le BIT n°6 qui à "1" correspond à une minuscule, et à "0" donne son équivalent majuscule. L'instruction du genre `Caractere = Caractere & 223;` transformera toute minuscule en majuscule. ATTENTION : Cette transformation n'est correcte que si le clavier utilisé retourne des codes ASCII et non de l'EBCDIC ou des codes clavier "IBM" qui sont totalement différents ...

0	1	1	0	0	1	0	1	e
0	1	0	0	0	1	0	1	E

Fig.21

Ne changer que le bit n°6 donc le Masque logique = 11011111 soit 223 en décimal.

```
if ((Caractere > 96) && (Caractere < 123)) Caractere = Caractere & 223;
    Caractere >= a      Caractere >= z      Caractere = Caractere - 32;
```

## > Application concrète des opérateurs de décalages.

Dans la procédure `void TRANSCODE()` on doit isoler le *nombre d'éléments* dans l'OCTET codé pour le Morse conformément à la Fig.17 donnée en page 11. Dans ce but on doit faire transiter les trois BITS de poids forts de gauche dans les trois BITS de poids faibles à droite. Par exemple en Fig.22 **A** on retrouve le codage du caractère **L** de la Fig.17 avec en violet le *nombre d'éléments*. On désire le déplacer à droite comme montré en **B** tout en forçant les autres BITS à

zéro. Dans ce but nous allons utiliser l'opérateur logique SHIFT de décalage à droite qui en C++ se code avec ">>". Par exemple l'instruction en `Nb_Elements = Lettre >> 1;` engendre le résultat montré en **C**. Tous les BITS ont été "poussés" d'une position à droite, celui de poids faible étant définitivement perdu. Sur la gauche un "0" est inséré. ATTENTION, c'est le cas pour un *byte* mais pour une variable signée (*De type char, int ou long.*) on insère le même état que celui

du BIT de gauche pour conserver le signe. Dans notre cas on désire décaler cinq fois, l'instruction utilisée dans **P01** devient `Nb_Elements = Lettre >> 5.` Un autre exemple se trouve dans l'instruction `if ((Lettre & 1) > 0) TRAIT(); else POINT();` qui combine **SHIFT** et *Masque logique*. Cette action est suivie de l'instruction `Lettre = Lettre >> 1;` qui utilise le masque logique 1 avec l'opérateur ET pour forcer dans le résultat tous les BITS à zéro sauf celui de poids faible et ainsi afficher un point ou un trait. Ensuite `Lettre = Lettre >> 1;` décale l'octet pour faire passer les éléments suivants d'une position à droite. Décalant l'OCTET représentatif de la gauche vers la droite, on traite ainsi le caractère de la droite vers la gauche ce qui était précisé en Fig.17 de la page 11. Sur un RESET de **P01**, si on a activé le Moniteur de l'IDE avec à 57600 baud le logiciel se présente, puis liste les lettres en morse sur l'écran et en impulsions sonores. Enfin, le programme saute à la boucle de base `void loop()` qui tourne à l'infini en faisant clignoter à 1Hz la LED bleue.

### 06) Chronométrer sans pénaliser la rapidité d'exécution du programme.

C'est dans la boucle de base `void loop()` que généralement on traite l'interface Homme/Machine et l'on souhaite que si l'opérateur frappe une commande, que cette dernière soit le plus rapidement possible prise en compte. On va supposer ici que c'est le cas même s'il n'y a pas encore de dialogue installé. C'est du reste pour contrôler le fonctionnement de cette boucle infinie que souvent je fais clignoter rapidement la LED Arduino en **D13**. Si le programme s'enlise dans une séquence "infinie", la LED ne clignote plus et le programmeur en est alors averti optiquement.

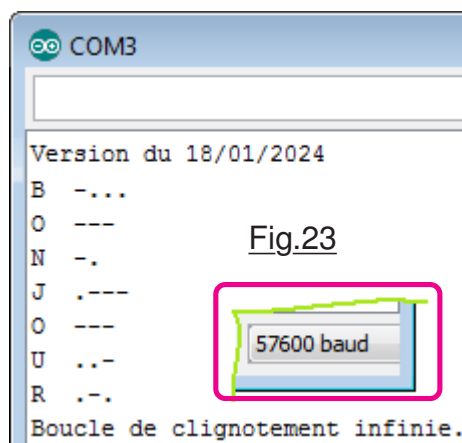


Fig.23

## ➤ Des chronomètres à gogo.

**C**haque usage de la fonction **millis** permet de créer un chronomètre indépendant qui ne bloque pas le déroulement du programme dans un **delay()** qui immobiliserait le microcontrôleur, et on peut en créer à volonté. Toutefois, chaque chronomètre devra se servir d'un "Top chrono" qui lui est propre. La Fig.24 présente un exemple avec deux chronomètres indépendants. Lorsque le microcontrôleur est mis sous tension ou lors d'un RESET, un compteur interne démarre à zéro et se voit incrémenté mille fois par seconde. (Il contient la durée écoulée en mS.) À tout moment **millis** peut en lire la valeur et la retourner sous forme d'un **unsigned long** en sortie de cette fonction. Pour mesurer l'intervalle de temps **T1** la valeur retournée par **millis** est enregistrée dans **Top\_1**. Puis dans le programme le chronomètre **A** va lire en permanence la valeur de **millis**. Dès que la valeur de **[millis - Top\_1]** sera égale ou plus grande que la valeur désirée **T1** c'est

que la durée calibrée sera atteinte et l'on déclenchera l'action pour **A**, puis immédiatement **Top\_1** sera rechargé avec **millis** pour démarrer un nouveau chronométrage. Pour réaliser un deuxième chronomètre **B** afin de mesurer une temporisation **T2** il suffit de dupliquer cette séquence, avec comme origine de chronométrage une mémoire **TOP\_2** et effectuer la comparaison **[millis - Top\_2]**. On peut ainsi en créer autant que nécessaire. Ces séquences peuvent se situer n'importe où dans le programme avec des durées **T1** et **T2** quelconques. *Noter que la durée de la temporisation ne sera pas strictement égale à celle calibrée, car il faut tenir compte du fait que l'écart de temps étant arrivé, le programme peut être occupé ailleurs.* (Par exemple tant qu'un B.P. est cliqué ...)

## ➤ Analyse du chronométrage pour les LEDs.

**E**xemple typique d'un chronométrage sans pénalité pour la rapidité du logiciel, la Fig.25 présente le listage de la séquence qui gère les LEDs dans le démonstrateur **P01** et qui est invoquée à chaque cycle de la boucle de base infinie **void Loop**. C'est dans la zone **1** colorée en rose pastel que sont effectués les tests pour le chronométrage n°1. Cette comparaison est très rapide et la sortie est immédiate tant que la durée attendue n'est pas atteinte. Puis, dès que l'on dépasse la temporisation consignée dans la valeur mise en évidence en jaune, on déclenche toutes les actions de la zone vert

```

void loop() {
  1 //===== Chronométrage pour le délai écoulé =====
    if (millis() - Temps_1_Ecoule_depuis_RESET > Delai_voulu) { Début
  2   Temps_1_Ecoule_depuis_RESET = millis(); // Réarmer le chronométrage. (TOP 1.)
    //----- Actions à traiter pour chaque délai désiré. -----
    Inverser_LED_bleue(); } Fin
  4 //===== Chronométrage pour le délai rapide =====
    if (millis() - Temps_2_Ecoule_depuis_RESET > 50) {
  5   Temps_2_Ecoule_depuis_RESET = millis(); // Réarmer le chronométrage. (TOP 2.)
    //----- Actions à traiter pour chaque délai écoulé. -----
    Inverser_LED_Arduino(); } }

```

Fig.25

3

6

pastel **3**. La première action urgente pour obtenir un chronométrage précis, est en **2** le **Top\_1** de la Fig.24 et dans notre cas il y a besoin de mémoriser cette valeur puisqu'il y a deux chronométrages à gérer. Généralement les actions **3** n'exigent que peu de cycles microcontrôleur et ensuite la boucle de base reprend sa routine de fond généralement à un rythme élevé. De façon totalement analogue on effectue dans la zone rose pastel **4** le chronométrage pour le clignotement rapide. Puis, dès que l'on dépasse la temporisation ici donnée en constante dans la zone jaune, on déclenche toutes les actions de la zone vert pastel **6**. Immédiatement c'est en **5** que le **Top\_2** est mémorisé. Noter que pour le délai de la LED bleue la valeur est paramétrée en tête de programme et facile à modifier.

**RÉSUMÉ : Pour réaliser un chronométrage on peut à convenance :**

- Utiliser l'instruction **delay()** qui "fige" le microcontrôleur durant tout le chronométrage,
- **Utiliser la fonction **millis** qui permet de créer des chronomètres indépendants,**
- De créer un chronométrage géré par le programmeur au moyen d'un simple compteur.

(On peut aussi faire par "interruptions", mais c'est un autre sujet !)



### ➤ Paramétrer en tête de programme.

Dans le démonstrateur **P01**, par exemple dans la procédure **void setup()** on doit modifier la date de validation du logiciel en question, ou dans ce même programme modifier la rapidité de la manipulation Morse virtuelle, ou encore la cadence de la transmission des données sur la ligne USB. Tout au long du développement d'un logiciel il peut y avoir besoin de modifier parfois des constantes ou de nombreux paramètres et on doit pouvoir facilement repérer les lignes d'instruction concernées. Il suffit de les faire terminer par des remarque de type `//@@@@@@@@@@@@@@@@@@@@`. Toutefois, quand on est amené à corriger des valeurs dans un listage bien plus "étalé" que celui du démonstrateur **P01**, *il est fortement recommandé de définir tous les paramètres d'initialisation en tête de programme et si possible de les regrouper*. Ainsi, on a une énumération compacte et l'on ne risque pas d'en oublier, et surtout c'est bien plus rapide que d'avoir à chercher de multiples emplacements dans un long listage. Cette façon de faire concerne directement la "programmation

```
//----- Constantes du programme -----  
#define Version "19/01/2024" // @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
#define Vitesse USB 57600 // Taux de transfert sur la ligne USB @@@@@@@@@@@@@@@  
#define Delai voulu 500 // Temporisation en mS @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
#define Rapide false // Cadence de la manipulation morse @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

Fig.26

avec méthode". Aussi, dans le programme de développement **P01\_Tester\_le\_materiel.ino** la date de version est définie en tête de listage par la constantes **Version** et le taux de transfert sur la ligne de dialogue USB est défini par la constante **Vitesse\_USB**. Du reste, comme on peut l'observer sur la Fig.26 on a regroupé quatre paramètres sous forme de déclarations **#define**. Dans ces dernière il est possible de définir des chaînes de caractères, des entiers, des réels, des booléens etc. Par exemple la valeur **false** pour **Rapide** engendrera une cadence lente pour la manipulation Morse virtuelle. Je vous invite à la remplacer par **true** pour voir la différence et remplacer le **500** par 1000 par exemple pour voir la différence.

### ➤ Rapidité de "rotation" de la boucle de base.

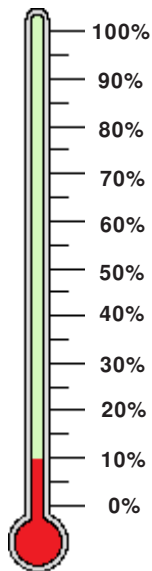
Dans les nombreux conseils prodigués péremptoirement dans ce didacticiel, il me semble important de préciser que l'on doit mesurer la rapidité d'exécution **void loop()** pour vérifier que la cadence de prise en compte des consignes opérateur soit élevée, donc vérifier que le *dispositif est réactif*. Seule limite, il faut disposer d'un fréquencemètre. Pour ceux qui le désirent, sur <https://www.robot-maker.com/ouvrages/apprendre-a-programmer-arduino-en-samusant/> est décrit ce type d'appareil réalisé avec une carte Arduino.

On obtient pour le démonstrateur **P01** une période de 102mS soit une fréquence de 9,79 Hz. La prise en compte des consignes arrivant sur la ligne USB sera donc rapide.

**NOTE IMPORTANTE :** Chaque fois que c'est possible dans un programme, je réserve la sortie **D13** pour gérer la LED Arduino dans la boucle de base. C'est généralement cette boucle de base qui "tourne" en tâche de fond et qui s'occupe du MENU de base du programme d'exploitation. Il importe que **void loop()** soit la plus rapide possible pour effectuer la prise en compte des divers claviers et codeurs rotatifs du système. Aussi il faut l'optimiser et en vérifier sa cadence. *En outre, le clignotement rapide de la LED Arduino atteste également du fonctionnement de void loop(). Aussi, si cette LED ne clignote plus, c'est que le programme s'est enlisé dans une séquence très longue, ou pire, dans une boucle infinie dont il ne sortira plus.*

### ➤ Un bilan de consommation des ressources.

Compilant **P01** on constate que le "Sketch" se goinfre déjà de 3128 Octets de l'espace réservé au programme soit 10% de ce dernier alors que l'on n'a encore écrit aucune routine destinée à ÉNIGMA. Nous sommes en droit de nous inquiéter et d'envisager l'impossibilité d'aboutir dans ce projet par manque de place pour le programme alors que pour ce petit démonstrateur le code a été optimisé. Pas de panique ! Une longue expérience en programmation C++ sur Arduino a montré que dans tous les programmes, les premières séquences sont boulimiques. Mais plus les sous-routines installées augmentent, plus l'inflation diminue, car le compilateur ensuite se contente d'invoquer les séquences installées. Par exemple dans notre démonstrateur la simple utilisation de la ligne série engloutit à elle seule environ 272 octets uniquement pour **Serial.println()**; Ensuite, les autres affichages deviennent bien moins gourmands. Par ailleurs, les textes sont considérés



comme des tableaux et prennent beaucoup de place à la fois dans la zone réservée au programme et empiètent de façon équivalente dans la zone réservée à la mémoire dynamique. Un moyen particulièrement efficace consiste à loger les textes du dialogue en mémoire non volatile EEPROM. Aussi, et même si ce n'est pas indispensable, quand le moment sera venu on ne va pas s'en priver, y compris si ce n'est pas utile. (*"Le moment sera venu" quand le logiciel contiendra suffisamment de textes pour que ce soit rentable.*) Par ailleurs on a déjà mis en place les séquences qui génèrent du code morse, et elles vont certainement être intégrées dans le programme final d'exploitation du petit boîtier bleu. Bref, on peut rester sereins et ne pas faire preuve de pessimisme et ce d'autant plus qu'actuellement les démonstrateurs sont compilés avec la version 1.7.9 de l'IDE. Hors la version 1.8 du compilateur est plus optimisée et le code qu'elle génère est bien moins volumineux. Donc si nécessaire on terminera notre cheminement avec cette version du compilateur de l'IDE.

(Personnellement je préfère la version 1.7.9, mais pour des raisons "futiles !")

Fig.27

Pour montrer de façon visuelle la progression de la taille du programme par rapport aux ressources disponibles, j'aime bien la représenter sous la forme d'un thermomètre avec en rouge le pourcentage actuellement occupé et en vert la place encore disponible. Chaque fois qu'une "fonction spécifique" aura été développée et validée, je reprendrais cet artifice visuel pour monter l'évolution du programme d'utilisation de notre Énigma virtuelle.

## 07) Protocole d'utilisation de cette machine ÉNIGMA.

Presque logiquement, on serait tenté de commencer à programmer les organes virtuels de la codeuse. Ce n'est pas du tout idéal, car il faut pouvoir tester les résultats obtenus. Comme le seul dispositif disponible est le **Moniteur** de l'IDE qui nous fournit sur le P.C un clavier et un écran, autant dès ce deuxième démonstrateur **P02\_Dialoguer\_avec\_Enigma.ino** établir les protocoles de l'interface Homme/Machine. Considérons donc, que comme montré sur la Fig.17 nous avons le petit coffret bleu réuni au P.C. par sa ligne série USB. On suppose évidemment qu'une version de l'environnement IDE d'Arduino est présente sur l'ordinateur.

### ➤ Architecture d'utilisation du système.

Dialogue Homme / machine oblige, on va utiliser dans notre cas un ordinateur de bureau ou un P.C. portable, ce qui en Fig.28 est le cas. On a branché sur la mini prise USB de la carte Arduino NANO en 1 la fiche 2. La ligne USB 3 est à son tour branchée sur l'une des prises de l'ordinateur 4. En 5 on a activé l'IDE qui a ouvert sa fenêtre contextuelle. En cliquant en 6 on invoque

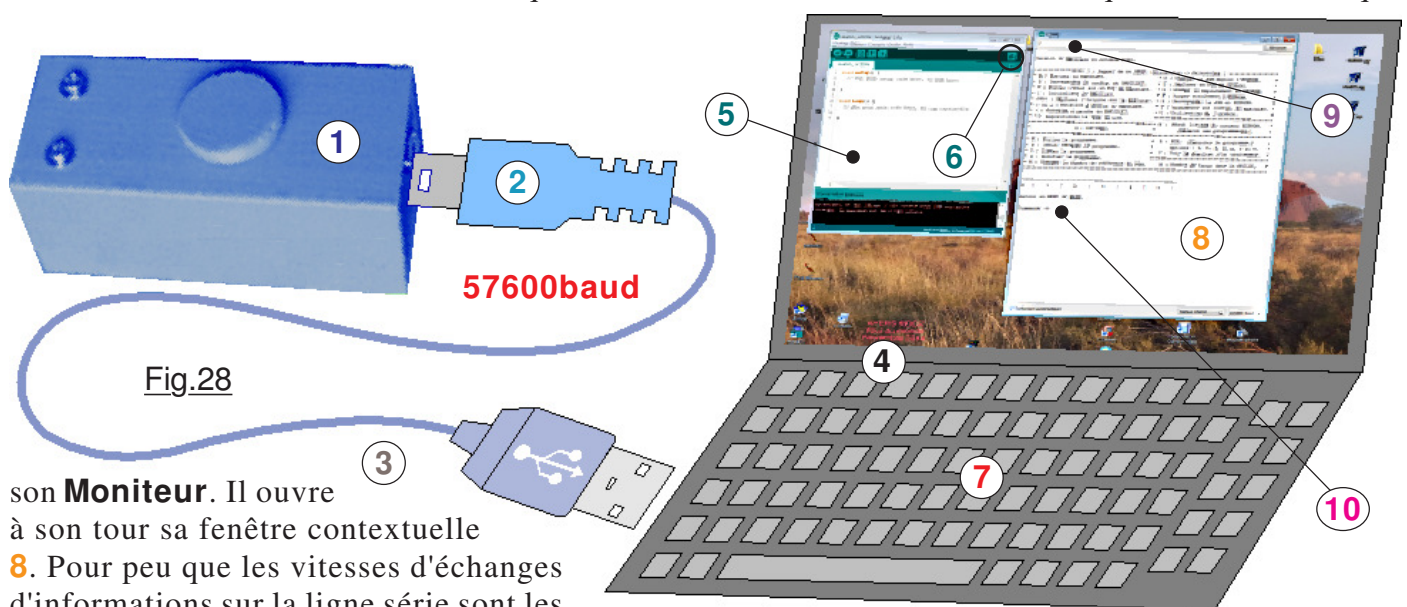


Fig.28

son **Moniteur**. Il ouvre à son tour sa fenêtre contextuelle 8. Pour peu que les vitesses d'échanges d'informations sur la ligne série sont les mêmes sur le P.C. et sur Arduino, apparaît alors le **MENU de base**. Dans la petite fenêtre de saisie 9 on visualise les commandes frappées sur le clavier 7. Le retour de la carte NANO 1 est alors affiché en 10 établissant ainsi de façon très simple "le dialogue Homme / Machine".

## ➤ Protocole d'utilisation du système.

Fondamentalement deux modes de fonctionnement "contradictaires" vont coexister sur notre codeuse. Soit on sera en mode **UTILISATION**, soit on naviguera en mode **COMMANDE**. Par construction de la machine Énigma, en mode **UTILISATION** on ne frappera que des caractères alphabétiques. Donc ***pour changer de mode on ne peut pas utiliser une lettre***. Après divers essais, le choix pour passer d'un mode à l'autre et ***dans les deux cas*** on se servira du caractère '&' facile à repérer sur le clavier et n'imposant pas l'obligation de "shifter".

*Quelle que soit la touche frappée on ne sera pas obligé de passer en MAJuscule.*

Pour le mode **COMMANDE**, à ce stade on envisage les options suivantes :

- **?** ou **,** : Rappel de ces commandes.
- **I** ou **i** : Passer en mode **INITIALISATION**.
- **S** ou **s** : Sauvegarder la configuration d'initialisation en EEPROM.
- **R** ou **r** : Restituer la configuration d'initialisation présente en EEPROM.
- **M** ou **m** : Traduire la lettre transcodée par la machine en ***code Morse sonore***. (*OUI/NON*)
- **T** ou **t** : Passer en mode **TEXTE**. (*Caractères envoyés par phrases et ESPACE accepté.*)
- **L** ou **l** : Passer en mode **LETTRES**. (*Caractères par caractère et affiché en morse.*)
- **&** ou **1** : Passer du mode **UTILISATION** au mode **COMMANDE** et réciproquement.

Pour le mode **INITIALISATION**, à ce stade il est prévu les options suivantes :

- **&** ou **1** : Retour au mode **COMMANDE**.
- **R** ou **r** suivi d'un chiffre : Choix du **Rotor** et de sa position sur le **Brouilleur**.
- **B** ou **b** et **'C'** ou **'c'** : Choix du **Rélecteur** installé sur le **Brouilleur**.
- **F** ou **f** suivi de deux lettres et d'un nombre de **1** à **10** : Choix de **Fiches croisées**.
- **E** ou **e** suivi de **'R'**, **'C'** ou **'F'** : Corriger une **Erreur**. (***C** pour le réflecteur de **Compléme**,t.*)
- **P** ou **p** indiquera l'espace disponible pour la **PILE**.

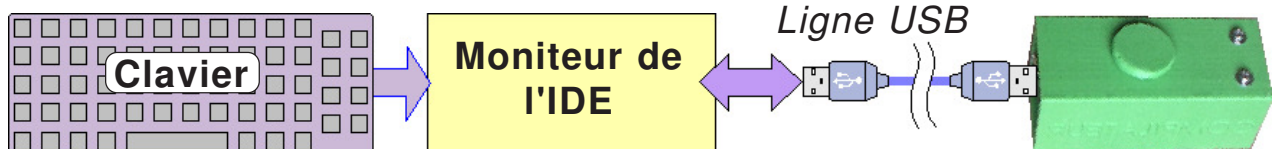
*Tout caractère invalide génèrera un BIP sonore d'erreur.*

Nous n'en sommes qu'au stade de la "prospective", c'est à dire qu'après analyse on a fait des choix qui semblent convenir. Il est évident que seule la pratique validera ces choix qui seront potentiellement plus ou moins remis en cause. L'utilisation des démonstrateurs peut aussi faire apparaître le besoin de nouvelles options. Bref, cette organisation n'est qu'une base de départ qui restera on s'en doute à peaufiner.

### 08) Le dialogue Homme / Machine.

Armerez-vous de patience car dans ce chapitre on ne va pas encore aborder l'agencement virtuel de l'un quelconque des organes de la machine. La raison en est fort simple. En effet lorsque l'on va émuler un **Rotor** par exemple, il nous faudra présenter au programme un caractère alphabétique, et voir comment les séquences élaborées le transforment. Le plus simple consiste à envoyer les consignes par l'entremise du **Moniteur** et réceptionner les accusés de réception sur ce dernier.

Fig.29



La chaîne de transmission est résumée sur la Fig.29 la fenêtre contextuelle du **Moniteur** étant affichée sur l'écran de l'ordinateur. Lorsque l'opérateur frappe un caractère au clavier, ce dernier est affiché dans le champs de saisie. Puis à la validation ce caractère ou le texte saisi est validé. Il est alors transmis à 57600baud sur la ligne USB. Capturé par le programme d'Arduino qui est en attente, ce texte ou ce caractère sont traités conformément au logiciel. Ce dernier retourne un accusé de réception qui est alors retourné sur la ligne USB. Le **Moniteur** réceptionne ce "compte rendu" et l'affiche dans la fenêtre de visualisation du **Moniteur**. Tous ces échanges sur la ligne USB sont simples à programmer avec l'instruction **Serial.println()**. Ce sont les protocoles de traitement du programme d'Arduino, c'est à dire le dialogue Homme / Machine qui vont être développés ici.



Par expérience, je peux m'engager à affirmer que c'est probablement la mise en œuvre de cette fonction de dialogue qui risque de conduire aux séquences les plus grandes en taille, car il va falloir établir du dialogue à la fois caractère par caractère, et aussi sous forme de textes. Généralement ce type de protocoles est assez glouton en octets, en particulier quand il faut extraire des caractères élémentaires (*Et en vérifier la validité.*) d'une chaîne textuelle. De plus, l'**Analyseur Syntaxique** des commandes s'avère généralement du genre un peu compliqué donc avec pas mal de cas particuliers à traiter. Enfin, les textes affichés pour l'opérateur sont nombreux et particulièrement boulimiques.


*Chaque fois que l'on désire établir un dialogue entre deux entités, on doit prendre en compte un maître et un esclave.* Le maître "parle" en envoyant une consigne à l'esclave qui "écoute". Puis le maître passe immédiatement en réception, et l'esclave réalise le travail imposé par la consigne. Puis il retourne un accusé de réception. Avec **P02\_Dialogue\_Homme\_Machine.ino** on va mettre en place l'interface d'utilisation. Lorsque le programme démarre, il affiche la référence de sa version suivi, comme montré sur la Fig.30, *des protocoles envisagés à ce stade du développement*. Ensuite nous pourrons commencer à créer l'Énigme virtuelle. Dans ces derniers on observe la commande '**P**'. Cette dernière sera explicitée plus avant dans l'un des derniers chapitres. Bien qu'elle soit émulée dans ce démonstrateur, pour le moment on peut royalement l'oublier. Pour se faire une idée du comportement que présentera le programme ultime, on va "se faire la main" avec le démonstrateur **P02** que vous téléversez dans les neurones de l'ATmega328. On va explorer les COMMANDES et le comportement en mode CRYPTAGE :

```

COM3
A
Fig.30
Version du 22/01/2024
*****
*           Menu des COMMANDES.           *
*****
* ? ou , : Rappel des commandes.          *
* I ou i : Mode INITIALISATION.           *
* S ou s : Sauvegarder la configuration.   *
* R ou r : Restituer la configuration.      *
* M ou m : Morse sonore OUI / NON.        *
* V ou v : Vitesse du débit du code Morse. *
* T ou t : Mode TEXTE.                    *
* L ou l : Mode LETTRES.                  *
* C ou c : Affiche la CONFIGURATION.       *
* P ou p : Espace disponible sous la PILE. *
* & ou 1 : Passer en Mode CRYPTAGE.        *
*****
*           Menu INITIALISATION.           *
*****
* R ou r : Choix du ROTOR et de sa position . *
* B ou b et C ou c : Choix du REFLECTEUR.   *
* F ou f : Configurer les FICHES croisees.  *
* G ou g : Groupes d'identification.         *
* E ou e suivi de R, C ou F : Corriger.     *
* (C pour Reflecteur de Complement.)        *
*****
*           CRYPTAGE en mode LETTRES.       *
*****
* & ou 1 : Passer en Mode COMMANDES.        *
* Frapper Uniquement des lettres et valider. *
*****
*           CRYPTAGE en mode TEXTE.         *
*****
* & ou 1 : Passer en Mode COMMANDES.        *
* Frapper des mots. (ESPACE valide.)        *
*****
Commande ->

```

## MANIPULATIONS :

- 01) Faire un RESET pour débiter sur une configuration "vierge de toute action préalable".
- 02) Cliquer sur  pour activer le Moniteur de l'IDE.
- 03) Si l'écran affiche des incohérences, c'est que la vitesse de transmission sur la ligne USB n'est pas à 57600 baud. Dans ce cas la modifier puis refaire un RESET.
- 04) Dans le champ de saisie **A** frapper la lettre de commande "**F**" puis valider.

- *Bande de Dudules, ce n'est pas une COMMANDE valide dont la liste est résumé dans la zone B.*  
- *Bon, on se concentre, et on continue avec allégresse !*

**NOTE :** Chaque fois que l'**Analyseur Syntaxique** des commandes détecte une erreur, il génère un texte d'alerte et un BIP sonore pour prévenir l'opérateur. Le caractère erroné est alors ignoré.

- 05) Toujours dans le champs de saisie **A** proposer la lettre de commande "**i**". *Elle est transformée en équivalent majuscule* et le programme précise que la fonction est traitée.

Dans ce démonstrateur les fonctions ne sont que titrées et ne font rien d'autre.

Naturellement on est autorisé à frapper les équivalents MAJuscules pour tous les caractères des protocoles adoptés. Néanmoins personnellement je trouve bien plus convivial de travailler exclusivement en minuscule, c'est à dire de ne pas avoir à "shifter". Donc, pour assurer la qualité opérationnelle du dialogue d'interface, les caractères valides frappés au clavier seront tous translatés en leurs équivalents majuscules comme précisé dans le MENU.

### MANIPULATIONS : (Suite)

- 06) Imposer les consignes des fonctions réciproques "s", valider puis "r" et valider. Dans les deux cas le démonstrateur précise le travail qui sera effectué quand ces fonctions seront émulées.
- 07) Pour voir ce qui va se passer frapper "sr" et valider cette paire de caractères tous deux valides.

|| En mode **COMMANDE** un seul caractère est autorisé. *Tous seront ignorés*, si on en propose plusieurs avant de valider. Pour en informer l'opérateur le programme génère un message d'erreur typé. (Par exemple ici les deux caractères étaient valides et sont non exécutés.)

- 08) Frapper alors un "&" pour passer en mode CRYPTAGE. Le changement de mode est signifié de façon non ambiguë. Maintenant seules des lettres sont autorisées ainsi que "&". Le texte d'invite "Commande ->" est remplacé par "Une lettre ->" précisant exactement ce qui est possible.
- 09) Proposer "a" et valider. Quand la machine Énigma sera simulée, chaque caractère sera "brouillé". Pour le moment le démonstrateur se contente de la transformer en sa suivante et cette dernière une fois modifiée est affichée en code Morse.
- 10) Revenir au mode COMMANDE avec '&'.  
11) Choisir l'option 'm' puis repasser en CRYPTAGE avec '&'.  
12) Maintenant frapper quelques caractères individuels. Ils sont traduits en Morse sonore et simultanément la LED bleue s'illumine à la cadence des traits et des points.  
13) Revenir au mode COMMANDE avec '&'.  
14) Changer la cadence avec 'v' qui passe en Vitesse Rapide.  
15) Retourner en CRYPTAGE avec '&'.  
16) Tester quelques caractères pour voir la différence.  
17) Reprendre le mode COMMANDE et museler le bruiteur avec 'm'.  
18) Modifier la cadence avec 'v'. (La rapidité s'inverse pour redevenir Lente.)  
19) Tenter à nouveau d'envoyer des caractères avec '&' suivi de quelques lettres.

|| La vitesse Lente a été mémorisée et le restera jusqu'à la prochaine commande de modification. Il en est de même pour l'option 'm' qui était mémorisée à "silence".

- 20) Expérimenter le message "bonjour" en ne validant qu'à la fin. Comme on a dépassé un caractère l'expérience se solde par un échec et aucune action n'est réalisée à part l'alerte.
- 21) Recommencer, mais en validant à chaque lettre. C'est vraiment laborieux. Aussi, on va changer radicalement de protocole en passant en mode TEXTE.

Pour augmenter l'agrément d'utilisation de cette machine virtuelle, le mode TEXTE nous permet de frapper directement des phrases dans la fenêtre de saisie A. Dans ce mode le texte crypté est présenté comme il l'était à l'époque de l'utilisation d'ÉNIGMA c'est à dire que les caractères sont cryptés et envoyés par groupe de six séparés par des espaces. Le mode TEXTE respecte ce formatage. Par ailleurs, pour rendre l'affichage plus rapide, en mode TEXTE la traduction en code Morse trop pénalisante en durée de transmission n'est plus active même si avec 'm' elle est mémorisée en 'OUI'. Naturellement elle sera retrouvée en mode LETTRES.

- 22) Imposer le mode COMMANDE avec '&' puis proposer 't'.

Noter que si dans la fenêtre de saisie on peut écrire autant de caractères que l'on veut, surtout si la fenêtre contextuelle prend toute la largeur de l'écran du moniteur vidéo, seul un maximum de 64 ne sera conservés dans la mémoire tampon de l'ordinateur lorsque l'on valide. C'est la raison pour laquelle le passage en mode TEXTE affiche "(60 caracteres MAXI.)". Du reste, c'est la raison pour laquelle je vous invite fortement à imposer à la fenêtre contextuelle du Moniteur de l'IDE la largeur minimale la plus adaptée pour notre application.

## MANIPULATIONS : (Suite)

- 23) Tester avec le texte "**a b c d e f g h i j k l m n o p q r s t**" précédé de '&'. On remarque que *les espaces ne provoquent plus d'erreur et sont ignorés*. Quand on codera des phrases on pourra ainsi intercaler (*Par la force de l'habitude.*) des espaces entre les mots sans pénalité.
- 24) Continuer la séquence avec "**u v w x y z**". (*Les caractères complètent le formatage.*)

Quand on passe **en mode TEXTE**, on suppose que **l'on va proposer un message complet à envoyer par radio**. Toutes les séquences valides frappées dans la fenêtre de saisie vont être cryptées "en continu" et compléter le formatage par groupe de six lettres.

- 25) Continuer avec "**bonjour les amis. Tout va bien.**". Le cryptage se poursuit sans incident. On constate que le '.' n'a pas engendré d'erreur. (*Comme pour l'espace le logiciel "pardonne."*)

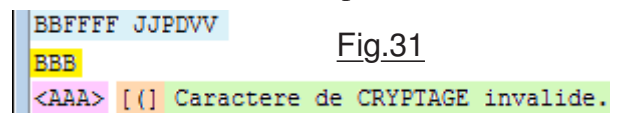
Quand un opérateur utilise une vraie ÉNIGMA, son clavier se limite à celui de la Fig.4B et il lui est impossible de proposer autre chose que des lettres. En ce qui nous concerne, on dispose d'un clavier d'ordinateur avec le risque par habitude de frapper des accentués. Aussi, pour faciliter les manipulations certains caractères spéciaux indiqués dans le tableau ci-contre sont transposés.

à devient a	î devient i
â devient a	ï devient i
é devient e	ô devient o
è devient e	ç devient c
ê devient e	û devient u
ë devient e	ü devient a

- 26) Expérimenter en proposant "**ââèèëëïïôçûü**". Les caractères sont tous acceptés.

- 27) Introduisons une erreur avec "**aaa(bbb**".

Fig.31



Considérons la Fig.31 dans laquelle en bleu pastel se trouve le cryptage précédent. Puis les caractères valides sont encodés dans la zone jaune. Un BIP sonore se fait entendre et entre crochets dans la zone orange le caractère incriminé est dénoncé suivi du texte vert. Le texte du message qui suit le caractère erroné est purement ignoré. (*Il peut contenir n'importe quoi y compris des erreurs.*) Pour que l'opérateur puisse continuer à crypter, le texte correct qui a été transcodé est précisé entre '<' et '>' non brouillé pour mémoire. Il suffit de reprendre la suite sans autre difficulté.

- 28) Pour finir cette prise en main de la machine, revenir aux COMMANDEs avec '&'.  
29) Enfin consigner la lettre 'p'.

**- Bande de Dudules, ce n'est pas le moment comme indiqué en début de ce chapitre !**

### ➤ Un bilan sur l'évolution du programme.

Nous n'avons toujours pas écrit une seule ligne pour commencer à réaliser notre codeuse virtuelle et déjà un cinquième de l'espace réservé au programme est consommé. Pourtant, à ce stade du développement il n'y a pas vraiment de raison de s'inquiéter. En effet, avec l'interface H/M je vous avais prévenu que le code serait probablement glouton en octets. De plus, les "dialogues" sont en place et l'analyse syntaxique est complète pour les fonctions actuelles. Par ailleurs on peut s'informer dans l'encadré situé en bas de la page 22 qui conseille que *si la mémoire non volatile EEPROM n'est pas utilisée pour mémoriser des données y logger un maximum de textes* affichés. C'est bien entendu ce que nous allons faire dans un démonstrateur futur. Sur le thermomètre est symbolisée en orange la place occupée par les textes affichés soit environ 5% de l'espace réservé au programme ce qui n'a rien de négligeable. Et surtout, ces textes considérés comme des tableaux sont clonés en mémoire dynamique dans laquelle ils consomment environ 849 octets sur les 2048 disponibles. Aussi, lorsque les textes seront passés en EEPROM ces 849 emplacements redeviendront disponibles pour les données temporaires. Il serait possible de procéder maintenant à cette refonte du programme. Mais si l'on ne commence pas à créer "du matériel", je sens que vous allez devenir fébriles ou nerveux. Aussi, dans le démonstrateur suivant on va débiter la création de la machine virtuelle en réalisant et en testant un **Rotor** virtuel. Il sera alors judicieux de le ranger en "magasin", c'est à dire en EEPROM nous en verrons la raison plus avant. On en profitera pour y logger tous les textes de l'interface H/M pour alléger le programme.



Fig.32



## 09) Le premier rotor de l'ÉNIGMA virtuelle.

Pour le démonstrateur **P03\_Premier\_Rotor.ino** on va se contenter de simuler la transformation quand le caractère circule de l'entrée à droite vers le **Réflexteur** à gauche. Puis on suppose que le **Réflexteur** le retourne sans le modifier pour émuler le comportement réciproque. Ainsi le caractère en sortie du "mouvement" de gauche vers la droite redevient celui initial. Il suffit de réutiliser **P02** et d'y introduire le **Rotor** et de s'en servir dans la procédure provisoire **Crypte\_le\_caractere()**. Le tableau de la Fig.33 résume les transpositions effectuées. **Fig.33**

<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>	<b>F</b>	<b>G</b>	<b>H</b>	<b>I</b>	<b>J</b>	<b>K</b>	<b>L</b>	<b>M</b>	<b>N</b>	<b>O</b>	<b>P</b>	<b>Q</b>	<b>R</b>	<b>S</b>	<b>T</b>	<b>U</b>	<b>V</b>	<b>W</b>	<b>X</b>	<b>Y</b>	<b>Z</b>	<b>Rotor I</b>
<b>E</b>	<b>K</b>	<b>M</b>	<b>F</b>	<b>L</b>	<b>G</b>	<b>D</b>	<b>Q</b>	<b>V</b>	<b>Z</b>	<b>N</b>	<b>T</b>	<b>O</b>	<b>W</b>	<b>Y</b>	<b>H</b>	<b>X</b>	<b>U</b>	<b>S</b>	<b>P</b>	<b>A</b>	<b>I</b>	<b>B</b>	<b>R</b>	<b>C</b>	<b>J</b>	Direct
5	2	13	6	12	7	4	17	22	26	14	20	15	23	25	8	24	21	19	16	1	9	2	18	3	10	Direct
<b>U</b>	<b>W</b>	<b>Y</b>	<b>G</b>	<b>A</b>	<b>D</b>	<b>F</b>	<b>P</b>	<b>V</b>	<b>Z</b>	<b>B</b>	<b>E</b>	<b>C</b>	<b>K</b>	<b>M</b>	<b>T</b>	<b>H</b>	<b>X</b>	<b>S</b>	<b>L</b>	<b>R</b>	<b>I</b>	<b>N</b>	<b>Q</b>	<b>O</b>	<b>J</b>	Réfléchi
21	23	25	7	1	4	6	16	22	26	2	5	3	11	13	20	8	24	19	12	18	9	14	17	15	10	Réfléchi

Par exemple **A** entrant devient **E** sur le **Réflexteur** dans le sens **Direct**.

Renvoyé par le **Réflexteur** dans le sens **Réfléchi** ce **E** redevient un **A**.

Lorsque **P03** est activé, rien n'interdit d'utiliser le mode TEXTE, mais je vous recommande dans un premier temps d'imposer le mode LETTRES. Si on frappe les caractères dans l'ordre alphabétique on obtient le résultat de la Fig.34 sur laquelle le caractère en entrée est dans la zone bleue et celui de sortie dans la colonne verte suivi en surface jaune par l'équivalent Morse. Sur le **Réflexteur** c'est la lettre dans la colonne rose qui est retourné de la gauche vers la droite. On teste ainsi le codage **Direct** avec le tableau **Rotor\_de\_DroiteD[26]** et le codage **Réfléchi** avec le tableau **Rotor\_de\_DroiteR[26]**. Pour la codeuse du programme ultime d'utilisation, il suffira de créer cinq **Rotors** de ce type, d'en choisir trois en fonction du jour du "conflit 39/45" et de les positionner sur la machine virtuelle.

Passage au mode CRYPTAGE.

Une lettre -> [A] devient [ <E> A ] .-  
 Une lettre -> [B] devient [ <K> B ] -...  
 Une lettre -> [C] devient [ <M> C ] -...  
 Une lettre -> [D] devient [ <F> D ] -..  
 Une lettre -> [E] devient [ <L> E ] .  
 Une lettre -> [F] devient [ <G> F ] ...-  
 Une lettre -> [G] devient [ <D> G ] --..  
 Une lettre -> [H] devient [ <Q> H ] ....

**Fig.34**

➤ **OUPS ... il manquait une commande !**

Désolé amis Internauts, mais quand j'ai réalisé le programme du dialogue utilisateur **P02** j'ai oublié une commande très importante. En effet, il est vital sur notre ÉNIGMA de pouvoir à tout moment vérifier sa configuration d'initialisation. C'est maintenant chose faite avec la commande '**C**' qui dans **P03** devient valide. Du coup elle est ajoutée dans la liste de la Fig.30 et repéré en gris car lors de l'utilisation de **P02** elle est encore interdite. Comme pour les autres fonctions non encore développées elle se contente de se nommer pour indiquer sa prise en compte.

➤ **Bilan relatif à la simulation d'un rotor.**

Quand on valide **P03** le compilateur annonce "**Le croquis utilise 7 058 octets (22%)**" soit à peine 180 octets de plus pour créer ce **Rotor** simulé. Autant dire presque rien, inutile de corriger le thermomètre représentatif de l'occupation de la mémoire réservée au programme, il ne change strictement pas. Pourtant, force est de constater qu'avec une si faible modification, le compilateur devient craintif et soucieux et sur la copie d'écran de la Fig.35 il affiche un message

Les variables globales utilisent 1 619 octets (79%) de mémoire dynamique, ce qui laisse 429 octets pour les variables locales. Le maximum est de 2 048 octets.  
 La mémoire disponible faible, des problèmes de stabilité pourraient survenir.

**Fig.35**

d'alerte en orange. En réalité ce n'est pas la taille du programme qui le préoccupe, j'ai déjà réalisé des logiciels qui occupaient 100% des octets réservés sans que ça ne pose problème. Son inquiétude vient du fait que trop de données sont logées dans la mémoire dynamique, il préférerait que la place occupée soit inférieure aux 79% actuels. Nous pourrions continuer à créer d'autres organes sur la machine, ne serait-ce que les quatre autres rotors. Hors ces derniers vont encore ajouter 208 octets dans la mémoire dynamique. Hors la solution consiste à placer ces tableaux en mémoire non volatile EEPROM comme précisé dans l'encadré rose en bas de la page 22. Aussi, pour éviter de créer ces **Rotors** dans le programme comme actuellement et avoir à Réécrire les routines

routines d'émulation avec stockage en EEPROM, je vous propose d'adopter la stratégie suivante :

- 1) On code les cinq **Rotors** tout en "haut" de la mémoire EEPROM.
- 2) On en profite pour logger à partir de l'adresse 0000 les textes qui seront utilisés de façon certaine.
- 3) On rédige la routine de rechargement d'un rotor. (*Et les procédures d'affichage des textes.*)
- 4) On teste dans **P04\_Les\_cinq\_Rotors.ino** les quatre autres **Rotors** avec la technique de **P03**.

Pour que vous puissiez comprendre le codage en C++ de ces **Rotors** le tableau de la Fig.36 en résume les permutations effectuées par les quatre que l'on va ajouter au n°1.

Fig.36

<b>A B C D E F G H I J K L M N O P Q R S T U V W X Y Z</b>	<b>Rotor II</b>
<b>A J D K S I R U X B L H W T M C Q G Z N P Y F V O E</b>	Direct
1 10 4 11 19 9 18 21 24 2 12 8 23 20 13 3 17 7 26 14 16 25 6 22 15 5	Direct
<b>A J P C Z W R L F B D K O T Y U Q G E N H X M I V S</b>	Réfléchi
1 10 16 3 26 23 18 12 6 2 4 11 15 20 25 21 17 7 6 14 8 24 13 9 22 19	Réfléchi

<b>A B C D E F G H I J K L M N O P Q R S T U V W X Y Z</b>	<b>Rotor III</b>
<b>B D F H J L C P R T X V Z N Y E I W G A K M U S Q O</b>	Direct
2 4 6 8 10 12 3 16 18 20 24 22 26 14 25 5 9 23 7 1 11 13 21 19 17 15	Direct
<b>T A G B P C S D Q E U F V N Z H Y I X J W L R K O M</b>	Réfléchi
20 1 7 2 16 3 19 4 17 5 21 6 22 14 26 8 25 9 24 10 23 12 18 11 15 13	Réfléchi

<b>A B C D E F G H I J K L M N O P Q R S T U V W X Y Z</b>	<b>Rotor IV</b>
<b>E S O V P Z J A Y Q U I R H X L N F T G K D C M W B</b>	Direct
5 19 15 22 16 26 10 1 25 17 21 9 18 8 24 12 14 6 20 7 11 4 3 13 23 2	Direct
<b>H Z W V A R T N L G U P X Q C E J M B S K D Y O I F</b>	Réfléchi
8 26 23 22 1 18 20 14 12 7 21 16 24 17 3 5 10 13 2 19 11 4 25 15 9 6	Réfléchi

<b>A B C D E F G H I J K L M N O P Q R S T U V W X Y Z</b>	<b>Rotor V</b>
<b>V Z B R G I T Y U P S D N H L X A W M J Q O F E C K</b>	Direct
22 26 2 18 7 9 20 25 21 16 19 4 14 8 12 24 1 23 13 10 17 15 6 5 3 11	Direct
<b>Q C Y L X W E N F T Z O S M V J U D K G I A R P H B</b>	Réfléchi
17 3 25 12 24 23 5 14 6 20 26 15 19 13 22 10 21 4 11 7 9 1 18 16 8 2	Réfléchi


L'implantation de l'EEPROM va respecter l'organisation de la Fig.37 avec dans les adresses hautes le codage des cinq **Rotors** possibles en premier et des deux **Réflexeurs** potentiels tout en haut. Pour les cinq **Rotors** il faut prévoir  $5 \times 2 \times 26 = 260$  emplacements. Pour les deux réflexeurs c'est  $2 \times 26 = 52$  cellules qu'il faut employer. La zone rose qui ne sera plus modifiée consomme donc 312 octets et va des adresses décimales **712 à 1023**. Puis, dans la zone bleue qui précède, commence en **666** et se termine à la cellule **711**, on logera la configuration initiale sauvegardée en mémoire non volatile. Elle sera de taille constante et à ce stade du projet l'organisation de cette donnée analysée consommera 46 OCTETS. Le développement le confirmera ou imposera une modification de l'implantation. Juste "en dessous" dans la région jaune on prévoit huit OCTETS


#### Optimisation de la taille des variables.

Dans le cadre de la **programmation méthodique**, nous avons vu en Page 9 qu'il est très conseillé lors de l'écriture du programme d'ordonner les variable par type, par taille et par ordre alphabétique. Par ailleurs, **il est absolument indispensable de choisir le type de chaque variable pour en minimiser la place occupée en mémoire dynamique et en accélérer le traitement**. C'est un impératif absolu. **Du bon choix du type des données dépend considérablement la taille occupée par le programme et le temps d'exécution**. **ATTENTION**, les tableaux sont les données les plus voraces en espace utilisé **et les textes sont intrinsèquement des tableaux**. Aussi, **si la mémoire non volatile EEPROM n'est pas utilisée pour mémoriser des données** à conserver sur coupure alimentation, **y logger un maximum de textes** affichés. Enfin, **pour une lisibilité maximale du listage, toujours choisir avec beaucoup d'attention les noms des identificateurs des constantes et des variables** avec des noms qui "parlent".

pour des données ou options qui ne sont pas encore définies. (*Développement futur.*) Enfin, en partant du bas de l'adresse relative **0000** et s'étendant vers le haut dans la région verte, on logera un maximum de textes. Cette zone est saturée. Elle a été optimisée. Les autres chaînes de caractères résideront dans le programme. Ce n'est pas idéal, mais nous n'avons pas le choix.

### ➤ Textes, Rotors et Réflecteurs en EEPROM.

Première étape avant de pouvoir utiliser les textes, et les éléments virtuels de la machine, il faut les inscrire dans la mémoire non volatile du microcontrôleur. Dans ce but, disponible mais non encore utilisé, on va mettre en œuvre **P00A\_Initialiser\_EEPROM.ino** prévu pour cette mission. La Fig.38 liste les textes qui sont possibles, car l'EEPROM est saturée et seulement une partie du dialogue H/M peut y être logé. La zone bleue est occupée par la sauvegarde de l'initialisation d'ÉNIGMA dont une étude préliminaire à ce stade du projet estime l'encombrement à 46 OCTETS. On commence par activer le **Moniteur** de l'**IDE** avec l'idéogramme  et on initialise

sa vitesse de transfert à **57600** baud. Puis on téléverse l'outil informatique **P00** que l'on active ensuite en cliquant une deuxième fois sur . La fenêtre du moniteur se remplit et ressemble à la copie d'écran de la Fig.38 avec en violet des commentaires au fur et à mesure que le programme se déroule. En particulier dans la zone verte les textes inscrits sont listés. Puis il y a affichage en Fig 39, sous forme de tableaux, de l'intégralité du contenu de l'EEPROM avec l'indication des **adresses** des OCTETS dans la zone orange. (*Adresse du premier octet de la ligne à gauche à laquelle on doit*

```
Contenu EEPROM du vendredi 26 Janvier 2024.
Ecriture en EEPROM des textes :
Rapide.Lente.Menu des COMMANDES.
Une lettre ->
? ou , : Rappel des commandes.
I ou i : Mode INITIALISATION.
S ou s : Sauvegarder la configuration.
R ou r : Restituer
M ou m : Morse sonore OUI / NON.
V ou v : Vitesse du débit du code Morse.
T ou t : Mode TEXTE.
L ou l LETTRES.
C ou c : Affiche la CONFIGURATION.
P ou p : Espace disponible sous la PILE.
& ou l : Passer en Mode CRYPTAGE.
R ou r : Choix du ROTOR et de sa position.
B ou b et C ou c : Choix du REFLECTEUR.
F ou f : Configurer les FICHES croisées.
G ou g : Groupes d'identification.
E ou e suivi de R, F ou G : ERREUR.
Attention : LGR > MAX. Caractère invalide.
] devient [ Passage au mode
Retour
Inscription des Rotors et des Reflecteurs.
```

Fig.38

ajouter la "valeur" de la colonne affichée ici en hexadécimal.) Noter qu'à tout moment dans la liste des logiciels fournis vous disposez à convenance de **Lister\_EEPROM\_en\_HEXAdecimal.ino** un outil informatique qui vous permet de lister le contenu de l'EEPROM de n'importe quelle carte Arduino.

Dans la zone verte de ce long listage formaté en matrice, sont donc listés en clair les caractères des mots, pour ainsi déterminer l'adresse de leur début indispensable aux routines d'affichage qui viendront puiser les éléments. On remarque au passage trois accentués, car leur codage en EEPROM ne pénalise plus la taille occupée par le code OBJET. La zone rose de la Fig.39 a été fragmentée pour repérer dans notre magasin les divers composants de la machine. Dans les secteurs violets et roses sont confinés les **Rotors** dans le sens direct et dans le sens réfléchi. Dans la zone marron clair on trouve le **Réflecteur B** et enfin dans l'encadré gris clair le **Réflecteur C**. Les divers éléments sont en place, il ne reste plus qu'à les exploiter dans le démonstrateur **P04**.



Fig.37

### Un petit outil qui peut toujours servir.

Durant la programmation des nombreux "outils" de validation, il m'est arrivé de "polluer" le contenu de l'EEPROM. Aussi, si à un moment donné vous soupçonnez qu'un incident a peut être détruit des données dans la mémoire non volatile, le croquis **Lister\_EEPROM\_en\_HEXAdecimal.ino** permet d'en vérifier son contenu sans avoir à faire appel à P00. Éventuellement dans ce petit programme il est possible de remplir l'EEPROM avec des "FF" en validant deux lignes passées en remarque. Il sera peu utile pour nous de nous en servir, mais c'est un outil de base dont je me sers régulièrement.



ADRS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F			
0000	R	a	p	i	d	e	.	L	e	n	t	e	.	M	e	n		0496	E S c r o i s é e s . G o
0016	u	d	e	s	C	O	M	M	A	N	D	E	S	.				0512	u g : G r o u p e s d '
0032	U	n	e	l	e	t	t	r	e	-	>	?						0528	i d e n t i f i c a t i o n .
0048	o	u	,	:	R	a	p	p	e	l	d	e						0544	E o u e s u i v i d e
0064	s	c	o	m	m	a	n	d	e	s	.	I	o					0560	R , F o u G : E R R
0080	u	i	:	M	o	d	e	I	N	I	T	I						0576	E U R . A t t e n t i o n :
0096	A	L	I	S	A	T	I	O	N	.	S	o	u					0592	L G R > M A X . C a r a c
0112	s	:	S	a	u	v	e	g	a	r	d	e	r					0608	t é r e i n v a l i d e . ]
0128	l	a	c	o	n	f	i	g	u	r	a	t	i	o	n			0624	d é v i e n t [ P a s s a g
0144	.	R	o	u	r	:	R	e	s	t	i							0640	e a u m o d e R e t o u r
0160	t	u	e	r	M	o	u	m	:	M	o							0656	.....
0176	r	s	e	s	o	n	o	r	e	O	U	I	/					0672	.....
0192	N	O	N	.	V	o	u	v	:	V								0688	.....
0208	i	t	e	s	s	e	d	u	d	é	b	i	t					0704	..... 05 11 13 06 12 07 04 17
0224	d	u	c	o	d	e	M	o	r	s	e	.	T					0720	22 26 14 20 15 23 25 08 24 21 19 16 01 09 02 18
0240	o	u	t	:	M	o	d	e	T	E	X							0736	03 10 21 23 25 07 01 04 06 16 22 26 02 05 03 11
0256	T	E	.	L	o	u	l	L	E	T	T	R	E					0752	13 20 08 24 19 12 18 09 14 17 15 10 01 10 04 11
0272	S	.	C	o	u	c	:	A	f	f	i							0768	19 09 18 21 24 02 12 08 23 20 13 03 17 07 26 14
0288	c	h	e	l	a	C	O	N	F	I	G	U	R	A				0784	16 25 06 22 15 05 01 10 16 03 26 23 18 12 06 02
0304	T	I	O	N	.	P	o	u	p	:	E							0800	04 11 15 20 25 21 17 07 06 14 08 24 13 09 22 19
0320	s	p	a	c	e	d	i	s	p	o	n	i	b	l	e			0816	02 04 06 08 10 12 03 16 18 20 24 22 26 14 25 05
0336	s	o	u	s	l	a	P	I	L	E	.							0832	09 23 07 01 11 13 21 19 17 15 20 01 07 02 16 03
0352	o	u	l	:	P	a	s	s	e	r	e							0848	19 04 17 05 21 06 22 14 26 08 25 09 24 10 23 12
0368	n	M	o	d	e	C	R	Y	P	T	A	G	E	.				0864	18 11 15 13 05 19 15 22 16 26 10 01 25 17 21 09
0384	R	o	u	r	:	C	h	o	i	x								0880	18 08 24 12 14 06 20 07 11 04 03 13 23 02 08 26
0400	d	u	R	O	T	O	R	e	t	d	e	s						0896	23 22 01 18 20 14 12 07 21 16 24 17 03 05 10 13
0416	a	p	o	s	i	t	i	o	n	.	B	o	u					0912	02 19 11 04 25 15 09 06 22 26 02 18 07 09 20 25
0432	b	e	t	C	o	u	c	:	C									0928	21 16 19 04 14 08 12 24 01 23 13 10 17 15 06 05
0448	h	o	i	x	d	u	R	E	F	L	E	C	T	E				0944	03 11 17 03 25 12 24 23 05 14 06 20 26 15 19 13
0464	U	R	.	F	o	u	f	:	C	o	n							0960	22 10 21 04 11 07 09 01 18 16 08 02 25 18 21 08
0480	f	i	g	u	r	e	r	l	e	s	F	I	C	H				0976	17 19 12 04 16 24 14 07 15 11 13 09 05 02 06 26
																		0992	03 23 22 10 01 20 06 22 16 10 09 01 15 25 05 04
																		1008	18 26 24 23 07 03 20 11 21 17 19 02 14 13 08 12

### ➤ Simplification de l'affichage des commandes.

Dans le but de restreindre un peu "les bavardages", car nous savons maintenant que de nombreux textes vont encore résider en mémoire de programme, l'affichage par la commande '?' de la Fig.30 a été diminué, car les deux zones bleu clair et jaune n'apportent pas vraiment de l'information pertinente. Elles sont donc éliminées dans **P04\_les cinq Rotors.ino**. Par ailleurs, le texte " **E ou e suivi de R, F ou G : Corriger.**" n'était pas idéal car la lettre 'E' choisie ne correspondait pas directement à la fonction réalisée par cette commande. C'est la raison pour laquelle le texte a été modifié en " **E ou e suivi de R, F ou G : ERREUR.**" ce qui correspond mieux et du coup fait passer la zone jaune des OCTETS disponibles à 10 emplacements. (*Gain d'un octet.*)

### ➤ La lisibilité d'un programme.


Finalité de ce petit projet : S'amuser avec Arduino tout en apprenant à **programmer avec méthodes**. Dans cette optique il me semble incontournable d'aborder le thème de la lisibilité d'un logiciel. En Page 9 ce thème a déjà été abordé et je vous invite avec insistance à relire le chapitre qui commence par la vérité biblique **la première qualité d'un logiciel quel qu'il soit est sa LISIBILITÉ**. Nous allons observer dans le démonstrateur **P04** quelques lignes de programme où la lisibilité a été prise en compte. Considérons une instruction du type **Aff\_TEXTE\_EEPROM(76,30)** utilisée pour le dialogue Homme/Machine. Lors du développement chaque fois que l'on désire modifier un texte, il faut le changer en EEPROM. Puis modifier les paramètres le concernant dans le programme. Pour retrouver la bonne instruction dans le fatras des **Aff\_TEXTE\_EEPROM** c'est une galère, car **76,30** n'est vraiment pas lisible. C'est la raison pour laquelle il est important de faire suivre ces instructions par des remarques de type **// " l ou i : Mode INITIALISATION.**" précisant le texte affiché par l'instruction.

### ➤ Vérifier les cinq rotors qui sont dans notre magasin virtuel.

Par magasin virtuel vous avez bien compris qu'il est fait référence à la mémoire EEPROM de l'ATmega328. C'est non seulement cinq éléments qu'il va falloir vérifier, mais également dans les deux sens de "traversée" du courant électrique. De ce fait le démonstrateur **P04** doit

permettre de choisir le **Rotor** à tester, mais également le sens **Direct** ou **Réfléchi**. Le mieux pour tester ces séquences vitales du futur programme d'utilisation reste encore à effectuer un jeu d'essais complet dont voici le protocole :

### MANIPULATIONS :

- 01) Faire un RESET et téléverser **P04\_Les\_cinq\_Rotors.ino**.
- 02) Cliquer sur  pour activer le **Moniteur** à 57600 baud.
- 03) Dans la fenêtre de saisie frapper '**1**' pour tester le **Rotor I**.
- 04) Imposer '**d**' pour tester dans le sens **Direct**.
- 05) Frapper un '**i**' pour **Initialiser la machine dans cette configuration de test**.
- 06) Frapper alors un "&" pour passer en mode CRYPTAGE.
- 07) Proposer les 26 lettres possibles en validant pour chacune. On obtient le résultat de la Fig.40 avec dans la colonne bleue les lettres "entrantes" et dans la colonne verte celles cryptées. Il faut donc comparer ces dernières avec celles du tableau de la Fig.33 de la page 21.
- 08) Revenir au mode COMMANDE avec '&'.
- 09) Modifier le sens de "traversée" avec '**r**' pour **Réfléchi**.
- 10) Retourner en CRYPTAGE avec '&'.
- 11) Reprendre les 26 lettres possibles et vérifier le résultat sur **les valeurs rouges** du tableau.
- 12) Retour au mode COMMANDE avec '&'.
- 13) Frapper '**2**' pour tester le **Rotor II**.
- 14) Revenir au sens de "traversée" **Direct** avec '**d**' et '**i**' pour Initialiser. (*On teste dans l'ordre.*)
- 15) Reprendre les 26 lettres possibles et vérifier le résultat dans le tableau de la Fig.36 de la page 22.
- 16) Touche '&', touche '**r**' suivie de '**i**' pour tester le sens **Réfléchi**.
- 17) Reprendre ces procédures pour les autres **Rotors** :
  - Touche '&' / '**3**' / '**d**' / '**i**' / '&' / 26 lettres / '&' pour tester le **Rotor III** dans le sens **Direct**.
  - Touche '&' / '**r**' / '**i**' / '&' / 26 lettres / '&' pour vérifier le **Rotor III** dans le sens **Réfléchi**.
  - Touche '&' / '**4**' / '**d**' / '**i**' / '&' / 26 lettres / '&' pour tester le **Rotor IV** dans le sens **Direct**.
  - Touche '&' / '**r**' / '**i**' / '&' / 26 lettres / '&' pour vérifier le **Rotor IV** dans le sens **Réfléchi**.
  - Touche '&' / '**5**' / '**d**' / '**i**' / '&' / 26 lettres / '&' pour tester le **Rotor V** dans le sens **Direct**.
  - Touche '&' / '**r**' / '**i**' / '&' / 26 lettres / '&' pour vérifier le **Rotor V** dans le sens **Réfléchi**.
- 18) Tester '**m**' deux fois pour vérifier les textes logés en EEPROM.
- 19) Tester '**v**' deux fois pour vérifier également leurs textes en EEPROM.
- 20) Tester '**p**' pour valider son texte. (*Explications plus avant.*)
- 21) Enfin frapper sur '**l**' et sur '**t**' toujours pour vérifier les textes non volatiles.

Noter que dans le Menu apparait la commande '**G**' non encore émulée. (*Étudiée plus avant.*)

### ➤ Un nouveau bilan sur l'évolution du programme.

**V**ous avez remarqué qu'en tête de listage des démonstrateurs, en remarque sont précisés les évolutions par thèmes de la taille du skech lors de ses modifications. Les informations précisées par le compilateur sont fonction de sa version raison pour laquelle cette dernière est précisée.

Compilateur vl.7.9 :	PGM	RAM dynamique
Avant modification :	6598 Octets (21%)	1381 Octets (67%)
Passage des textes en EEPROM :	6036 Octets (19%)	517 Octets (25%)
Ajout 5 Rotors et 2 Réflecteurs :	6856 Octets (22%)	643 Octets (31%)

Fig.41

On constate que le fait de loger un maximum de textes en EEPROM a fait diminuer la taille du programme de 562 octets ce qui n'est pas du tout négligeable alors que simultanément les routines pour échanger des données avec l'EEPROM ont été ajoutées. Surtout, c'est la mémoire dynamique qui s'est allégée de 656 octets correspondant aux textes logés en EEPROM auxquels s'ajoutent 208 octets relatifs à la simplification de l'affichage du menu.

Par ailleurs, avec les essais effectués dans les manipulations, nous avons vérifié le comportement des cinq rotors, ainsi que tous les textes logés en EEPROM soit presque l'intégralité de cette dernière. On va pouvoir maintenant sereinement créer les deux **Réflecteurs** virtuels.

## 10) Les deux Réflecteurs de l'ÉNIGMA virtuelle.

Bien que la gestion d'un **Réflecteur** va ressembler indubitablement à celle d'un **Rotor**, elle demeure plus simple car il n'y a qu'un seul sens de "traversée". Sur la chiffreuse que l'on va simuler, c'est à dire celle **modèle M3 qui était utilisée dans la Kriegsmarine**, on disposait de deux **Réflecteurs B** et **C** qui pouvaient prendre place à gauche du **Brouilleur**. La Fig.42 précise les transpositions qu'ils effectuaient ce que devra simuler notre programme.

Fig.42

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	Reflectr. B
Y	R	U	H	Q	S	L	D	P	X	N	G	O	K	M	I	E	B	F	Z	C	W	V	J	A	T	Sortie
25	18	21	8	17	19	12	4	16	24	14	7	15	11	13	9	5	2	6	26	3	23	22	10	1	20	Sortie

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	Reflectr. C
F	V	P	J	I	A	O	Y	E	D	R	Z	X	W	G	C	T	K	U	Q	S	B	N	M	H	L	Sortie
6	22	16	10	9	1	15	25	5	4	18	26	24	23	7	3	20	11	21	17	19	2	14	13	8	12	Sortie

Il n'est pas utile de créer un autre démonstrateur pour tester ces deux réflecteurs, car ils sont déjà intégrés dans **P04\_Les\_cinq\_Rotors.ino** présente l'ossature pour assurer cette tâche. Il suffit de rajouter les deux commandes 'B' et 'C' pour choisir ces éléments. Les sens déjà initialisés seront ignorés pour ces éléments puisqu'une seule permutation sera opérée dans le **Réflecteur**.


### MANIPULATIONS :

- 1) Pour la forme "repartir à zéro" avec un RESET.
- 2) Dans la fenêtre de saisie frapper '6' pour imposer le **Réflecteur B**.
- 3) Frapper un 'i' pour **Initialiser la machine dans cette configuration de test.**
- 4) Frapper alors un "&" pour passer en mode CRYPTAGE.
- 5) Proposer les 26 lettres possibles en validant pour chacune. Comparer les résultats obtenus avec ceux du tableau "haut" de la Fig.42 le chiffrement étant en bleu.
- 6) Frapper alors un "&" pour revenir au mode COMMANDE.
- 7) Enfin proposer '7' pour vérifier le **Réflecteur C** suivi de 'i' pour le configurer.
- 8) Retourner en CRYPTAGE avec '&'. et proposer à nouveau 26 lettres en validant pour chacune.
- 9) Comparer les résultats obtenus avec ceux du tableau "bas" de la Fig.42 chiffrement en bleu.
- 11) Un dernier "&" pour laisser le démonstrateur en mode COMMANDE.

### > Initialiser la chiffreuse Enigma.

Pour le passage obligé avant de pouvoir "câbler virtuellement la codeuse" il nous faut impérativement disposer du tableau des Fiches croisées, du coup on va devoir mettre en place toutes les variables décrivant la structure de la machine. (*Variables précisant les rotors installés et leur indexations internes ...*) C'est **P05\_Initialiser\_la\_chiffreuse\_Enigma.ino** qui va mettre en place les séquences et les données afférentes à la nouvelle fonction. Comme ce démonstrateur ne contiendra plus que du code "définitif", il servira de noyau de base pour les développements futurs. On se contentera étape par étape de le reprendre pour ajouter pas à pas les nouvelles fonctionnalités. Testons l'initialisation qui se contente ici de conditionner les trois **Rotors** et les deux **Réflecteurs**.

### MANIPULATIONS :

- 1) Faire un RESET et téléverser **P05\_Initialiser\_la\_chiffreuse\_Enigma.ino**
- 2) Cliquer sur  pour activer le **Moniteur** à 57600 baud.
- 3) Dans la fenêtre de saisie frapper 'I' pour Initialiser la machine. Le programme affiche le mode ">>> INITIALISATION <<<" et demande le numéro du **Rotor** de **droite**.
- 4) Répondre avec le chiffre '4' par exemple. (*On a sélectionné le Rotor IV.*) Le programme enchaîne automatiquement la saisie pour le **Rotor** du **centre**.
- 5) Proposer le chiffre '2' par exemple. **P05** demande de saisir le **Rotor** de **gauche**.
- 6) Frapper alors un "9" pour générer volontairement une erreur. **Tant que la réponse ne sera pas correcte c'est à dire un chiffre compris entre 1 et 5 le programme insistera lourdement.**
- 7) Tester avec "3ABCD" pour voir la réaction du programme. (**P05 persiste**)



- 08) Tester avec **"333"** pour titiller l'analyseur syntaxique. Le programme se contente de signaler l'erreur. Toutefois, comme tous les caractères sont des chiffres valides, la réponse est prise en compte. (*Pour ne pas pénaliser l'opérateur si le clavier est passé inopinément en répétition.*)

### Indexation interne des rotors.

**L**orsque l'on installe l'un des cinq Rotors disponibles sur la machine, il faut au préalable modifier l'indexation interne, c'est à dire la correspondance entre les lettres visualisées et le brouillage interne.

*(Un peu comme un cadenas à chiffres ou un Cryptex à lettres.)*

Hors cette configuration interne est indiquée par les nombres allant de 1 à 26 dans les tables secrètes d'initialisations qui étaient fournies pour un ou plusieurs mois aux différents services durant le conflit. Cette position de la bague se trouve dans la colonne "Ringstellung". Sur le simulateur d'ÉNIGMA que je vous propose la configuration est en Fig.43 dans l'encadré jaune, à la fois en lettre ou en son nombre équivalent. (*Ordre de la lettre dans l'alphabet.*) Il existe sur la toile une kyrielle de simulateurs tous aussi bien les uns que les autres. Personnellement ma préférence va sur celui-ci :

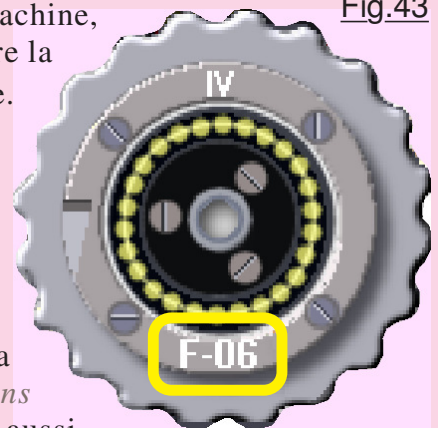


Fig.43

<https://www.ciphermachinesandcryptology.com/en/enigmasim.htm>

- 09) Proposer **"12"** pour l'indexation interne du **Rotor** placé à droite sur la machine.  
10) Puis **"26"** pour celui situé au centre sur la machine.  
11) Tester **"46"**, **'0'**, **'a'**, **"27"** pour constater qu'il faut impérativement donner des valeurs comprises entre 1 et 26 pour franchir cette étape.  
12) Enfin **"00008"** est accepté et montre que les zéros en tête sont ignorés.

Il faut encore refermer le couvercle de la machine quand les trois rotors sont en place, (*Voir la Fig.44*) et initialiser le chiffrement initial, c'est à dire l'orientation des trois rotors sous la fenêtre.



Fig.44

- 13) Frapper **'b'** pour orienter le **Rotor** placé à droite.  
14) Frapper **'x'** pour orienter le **Rotor** placé au centre.  
15) Tenter **'5'**, **'&'**, **'\*'** pour constater que dans cette phase tout caractère autre qu'une lettre sera ignoré et déclenchera un message d'alerte typé accompagné d'un BIP sonore.  
16) Tester **'é'** : **Les accentués sont acceptés et transformés en caractères simples**. À ce stade le programme enchaîne automatiquement sur la saisie pour le choix du **Rélecteur**.  
17) Tenter **'a'**, **'&'**, **'\*'**, **'5'**, **'d'**, **'f'** : Tout caractère autre que **'b'** ou **'c'** sera refusé.  
18) Vérifier avec **"bc"** : Il y a un message d'alerte, mais le premier caractère étant valide, il est correctement pris en compte et l'on revient au mode COMMANDE.  
19) Enfin proposer **'c'** pour faire afficher la configuration actuelle. On remarque que la présentation de la configuration est formatée et que le numéro des rotors est indiqué en chiffres romains.

### ➤ Encore un petit bilan sur l'évolution du programme.

**A**vec l'implantation de la fonction INITIALISATION nous avons amorcé une fonction particulièrement gourmande en octets. Rien d'étrange à cela, car elle enchaîne pas mal de saisies différentes qu'il faut accompagner avec des textes d'invite sans compter les nombreux tests à effectuer dans l'analyseur syntaxique pour détecter les erreurs de logique dans les manipulations. Ces derniers sont forcément gros consommateurs d'énergie. Même remarque pour l'affichage formaté de l'initialisation de la machine. Actuellement, cette fonction consomme environ 1454 octets de programme soit 5% de l'espace disponible. Ça n'a rien d'anormal et il ne faut pas s'en inquiéter même si les dialogues ont augmenté la place prise dans la mémoire dynamique de 6% environ. Nous savons que les textes des dialogues de l'interface Homme/Machine impactent directement les ressources de l'ATmega328. C'est la raison pour laquelle il importe de les réduire au maximum tout en conservant la convivialité d'exploitation ... un compromis délicat à satisfaire !

## ➤ Changement de stratégie, simplification du menu de base.

Initialiser la machine "en cascade" s'avère bien plus agréable à l'usage que d'avoir à spécifier à chaque élément à conditionner son type. C'est donc après diverses tentatives la ligne de conduite qui a été adoptée. Les manipulations sont donc plus conviviales ce qui améliore la qualité opérationnelle de notre chiffreuse virtuelle. Du coup, le sous-menu vert de la Fig.30 en page 18 ne se justifie plus ce qui permet une simplification du programme et un gain de place important dans la mémoire dynamique. (*Le nouveau menu est montré en Fig.45*) La contrepartie, c'est qu'il faut remplacer un certain nombre de textes dans l'EEPROM. Pour ne pas perturber le didacticiel, c'est dans une deuxième version de l'outil nommé **P00B\_Initialiser\_EEPROM.ino** que sont logés les "textes définitifs". Avant de procéder à ce travail assez rébarbatif, dans un démonstrateur provisoire ont été écrites un certain nombre de procédures futures. Ainsi on va placer dans l'EEPROM des textes "définitifs" et optimisés.

Par ailleurs l'implantation de la sauvegarde de la CONFIGURATION en mémoire EEPROM est maintenant stabilisée à celle de la Fig.46 où l'on voit qu'il ne reste plus que deux octets non utilisés

```
*****
*                               Menu des COMMANDES.                               *
*****
* ? ou , : Rappel des commandes.                               Fig.45 *
* I ou i : Mode INITIALISATION.                               *
* E ou e suivi de R, F ou G : ERREUR.                         *
* S ou s : SAUVEGARDER la configuration.                       *
* R ou r : RESTITUER la configuration.                         *
* M ou m : Morse sonore OUI / NON.                             *
* V ou v : Vitesse du débit du code Morse.                   *
* T ou t : Mode TEXTE.                                         *
* L ou l : Mode LETTRES.                                       *
* C ou c : Affiche la CONFIGURATION.                           *
* P ou p : Espace disponible sous la PILE.                   *
* G ou g : GROUPE d'identification.                             *
* & ou l : Passer en Mode CRYPTAGE.                             *
*****
```

ADRS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0000	R	a	p	i	d	e	.	L	e	n	t	e	.	M	e	n
0016	u	d	e	s	C	O	M	M	A	N	D	E	S	.		
0032	U	n	e	l	e	t	t	r	e	-	>	?				
0048	o	u	,	:	R	a	p	p	e	l	d	e				
0064	s	c	o	m	m	a	n	d	e	s	.	I	o			
0080	u	i	:	M	o	d	e	I	N	I	T	I				
0096	A	L	I	S	A	T	I	O	N	.	S	o	u			
0112	s	:	S	a	u	v	e	g	a	r	d	e	r			
0128	l	a	c	o	n	f	i	g	u	r	a	t	i	o	n	
0144	.	R	o	u	r	:	R	e	s	t	i					
0160	t	u	e	r	M	o	u	m	:	M	o					
0176	r	s	e	s	o	n	o	r	e	O	U	I	/			
0192	N	O	N	.	V	o	u	v	:	V						
0208	i	t	e	s	s	e	d	u	d	é	b	i	t			
0224	d	u	c	o	d	e	M	o	r	s	e	.	T			
0240	o	u	t	:	M	o	d	e	T	E	X					
0256	T	E	.	L	o	u	l	L	E	T	T	R	E			
0272	S	.	C	o	u	c	:	A	f	f	i					
0288	c	h	e	l	a	C	O	N	F	I	G	U	R	A		
0304	T	I	O	N	.	P	o	u	p	:	E					
0320	s	p	a	c	e	d	i	s	p	o	n	i	b	l	e	
0336	s	o	u	s	l	a	P	I	L	E	.	&				
0352	o	u	l	:	P	a	s	s	e	r	e					
0368	n	M	o	d	e	C	R	Y	P	T	A	G	E	.		
0384	G	R	O	U	P	E	d	'	i	d	e	n	t	i	f	
0400	i	c	a	t	i	o	n	d	e	.	d	e	D	r	o	
0416	i	t	e	d	e	G	a	u	c	h	e	d	u			
0432	C	e	n	t	r	e	V	e	r	s	i	o	n			
0448	d	u	O	r	i	e	n	t	a	t	i	o	n	R	O	
0464	T	O	R	E	F	L	E	C	T	E	U	R	d	é	j	
0480	a	u	t	i	l	i	s	é	e	.	F	o	u			

0496	f	:	C	o	n	f	i	g	u	r	e	r				
0512	l	e	s	F	I	C	H	E	S	c	r	o	i	s		
0528	é	e	s	.	E	o	u	e	s	u	i	v				
0544	i	d	e	R	,	F	o	u	G	:						
0560	E	R	R	E	U	R	.	A	t	t	e	n	t	i	o	
0576	n	:	L	G	R	>	M	A	X	.	U					
0592	n	s	e	u	l	C	a	r	a	c	t	è	r	e		
0608	!	i	n	v	a	l	i	d	e	.	]d	e	v			
0624	i	e	n	t	[	P	a	s	s	a	g	e	a			
0640	u	m	o	d	e	R	e	t	o	u	r	n	°			
0656	I	n	d	e	x	O	p	t	i	o	n	...	...			
0672	...	...	...	...	...	...	...	...	...	...	...	...	...			
0688	...	...	...	...	...	...	...	...	...	...	...	...	...			
0704	...	...	...	...	...	...	...	...	...	...	...	...	...			
0720	22	26	14	20	15	23	25	08	24	21	19	16	01	09	02	18
0736	03	10	21	23	25	07	01	04	06	16	22	26	02	05	03	11
0752	13	20	24	19	14	15	09	14	15	01	04	11				
0768	19	09	21	22	12	28	23	03	13	17	06	26	14			
0784	16	25	06	22	15	05	01	10	16	03	26	23	18	12	06	02
0800	04	11	15	20	25	21	17	07	06	14	08	24	13	09	22	19
0816	02	04	06	08	10	12	03	16	18	20	24	22	26	14	25	05
0832	09	23	07	01	11	13	21	19	17	15	20	01	07	02	16	03
0848	19	04	17	05	21	06	22	14	26	08	25	09	24	10	23	12
0864	18	11	15	13	05	19	15	22	16	26	10	01	25	17	21	09
0880	18	08	24	12	14	06	20	07	11	04	03	13	23	02	08	26
0896	23	22	01	18	20	14	12	07	21	16	24	17	03	05	10	13
0912	02	19	11	04	25	15	09	06	22	26	02	18	07	09	20	25
0928	21	16	19	04	14	08	12	13	10	17	15	06	05			
0944	03	11	17	03	25	12	24	06	20	26	15	19	13			
0960	22	10	21	04	11	07	09	01	18	16	08	02	25	18	21	08
0976	17	19	12	04	16	24	14	07	15	11	13	09	05	02	06	26
0992	03	23	22	10	01	20	06	22	16	10	09	01	15	25	05	04
1008	18	26	24	23	07	03	20	11	21	17	19	02	14	13	08	12

Fig.46

dans la zone jaune. En 5 sera placé **B** ou **C** du **Réflexeur**. En 1 sont indiqués les **Rotors** utilisés avec en 2 leur indexation interne et en 3 leur Orientation initiale. C'est en 4 que se loge le **GROUPE d'identification** du message transmis par l'opérateur radio. Enfin c'est en 6 que l'on préserve la combinatoire des dix **Fiches croisées** noté 5 sur la Fig.2 donnée en page 2.



## 11) Les protocoles de l'utilisation d'ÉNIGMA.

Opérateur radio dans l'équipage du U143 votre bâtiment navigue dans la zone BF21 proche des eaux anglaises. Chaque fois qu'un Uboat partait en mer, son commandant recevait autant de feuilles de codage du type de celle proposée en Fig.47 par mois qu'il devait rester en patrouille. Nous sommes le **12 février 1942**. Il faut INITIALISER la chiffreuse du bord pour la journée. Dans la colonne **UKW** est indiquée le **Réflecteur** à utiliser et dans **Walzenlage** l'ordre des **Rotors** à installer sur la machine avec dans la colonne **Ringstellung** leur Indexation interne. C'est dans **Steckerverbindungen** qu'est listée la combinatoire des **FICHES croisées**. Enfin, et c'est le but de ce chapitre, on trouve un **GROUPE d'Identification** dans la colonne **Kennggruppen**.

GEHEIM!

SONDER-MASCHINENSCHLUSSEL : FEVRIER 1942

FEBRUAR 42

Tag	UKW	walzenlage				Ringstellung			steckerverbindungen										Kennggruppen			
29	C	III	I	II		17	19	15	BG	CT	EM	FQ	HY	IR	KW	NS	PV	UZ	KMP	MQI	TEG	QUL
28	C	I	II	V		23	25	20	BQ	CV	DG	EI	JO	KP	LY	NW	RT	XZ	JEQ	TLU	LNS	PMP
27	B	I	II	V		07	15	21	BQ	DU	ES	FY	HN	IX	JV	KT	MZ	OW	UXG	IVJ	GDC	URC
26	C	III	I	V		24	15	09	AH	BT	CD	GI	JX	KL	NQ	PV	RZ	SU	CSU	RXA	WOA	QST
25	C	III	II	I		08	10	11	AV	CF	DK	EW	GM	IL	NQ	PU	RT	YZ	ZNP	KJU	OJZ	WWY
24	B	III	V	II		04	17	13	AL	CG	DN	ES	HP	IM	OW	RT	UZ	XY	JDH	MEE	ZEA	WEV
23	C	IV	V	II		05	20	09	AZ	BV	CG	EX	FO	HM	IW	KT	LR	PS	OWS	VTP	UVZ	IQN
22	C	V	I	IV		24	26	20	BT	CK	DI	FG	JO	MR	PZ	QY	SU	VW	YTM	STX	KZA	FLF
21	C	V	II	III		10	01	25	AR	BZ	DQ	FT	GI	HW	KP	LY	MV	UX	BSX	JOJ	CXD	HLN
20	B	I	IV	V		20	01	11	BN	CS	DU	EG	FM	HR	KO	LQ	PX	TY	VYX	WHJ	QBX	VTF
19	B	V	III	I		12	03		AD	BF	CW	EV	GN	IM	KL	QT	RY	SX	QKV	JPS	WTD	ILA
18	C	V	II	I		19	24	03	AL	BI	CG	EK	FN	MQ	OV	PU	SZ	WX	NTH	UAF	FEK	TXK
17	C	III	IV	II		09	16	09	CL	EW	FY	GR	IT	JU	KQ	MO	NX	PV	KHH	CGM	CXN	SGY
16	C	II	IV	III		18	03	02	BE	CK	DF	GT	HJ	IX	MN	OP	UW	VZ	DGL	DIJ	XII	ETO
15	B	I	V	IV		15	21	05	AH	BY	CQ	DK	EL	GZ	MS	NW	OR	UX	JCP	PYY	KKC	FOF
14	B	II	IV	I		13	25	25	BZ	CQ	DP	ET	FG	HI	JK	MW	RV	UY	VFD	MKP	ZAW	HZI
13	B	I	III	V		07	17	26	BT	DJ	EO	FS	GX	HP	KZ	MQ	UY	VW	WNL	HNB	MJX	LKU
12	B	IV	V	III		26	23	07	BJ	CQ	DI	FM	GO	HR	KW	PY	TU	XZ	DLP	IWQ	GWY	OAD
11	C	I	II	V		08	13	11	AM	BX	CT	EV	FY	GO	IN	JP	KQ	WZ	NNF	DID	KRR	ROA
10	C	V	II	I		20	11	02	AF	BY	CX	DO	ET	GL	JU	NZ	RS	VW	RUO	NRI	JSJ	PRQ
09	B	I	IV	III		21	06	13	AM	BO	CU	EX	GW	IL	JV	PQ	SZ	TY	PMF	XHG	RIX	YBB
08	B	V	I	II		26	14	14	DN	EP	FU	GX	HI	LT	MW	QS	RY	VZ	WWM	QIH	VZM	NLO
07	C	V	II	III		08	23	17	AQ	BU	DF	EJ	GT	HN	IP	KM	WZ	XY	TLW	ZSL	FCF	YGX
06	B	II	I	V		10	12	20	BP	CI	EK	FJ	GW	HQ	LX	MT	SZ	VY	BKL	MPN	DJE	HRO
05	C	IV	V	III		17	10	17	DU	EY	FI	HP	JR	KW	MX	NT	OQ	SZ	HWN	VYH	KEA	YYC
04	C	III	II	IV		21	07	06	AH	BQ	DR	EG	FX	IN	JY	LV	OU	SW	JHF	SDA	WKK	IZB
03	B	III	IV	I		08	22	10	CV	DS	EX	FP	GL	IZ	JW	KT	MU	NR	RBO	JNB	JAT	FPJ
02	C	IV	III	II		03	05	04	AJ	BE	CT	FS	GH	IM	KX	LY	NV	PU	RLO	KUQ	CZC	BED
01	C	II	IV	V		14	01	07	AE	CV	DH	FU	GK	IT	LX	MR	OW	SY	KHD	XTE	ONP	YTU

Concrètement ce groupe de 12 lettres avait un codage qui dépendait des services à qui le message était destiné et était fonction de l'époque, car il a changé au cours du conflit. Peu importe l'historique, on va se contenter de savoir qu'il était envoyé sans être chiffré par Énigma et surtout que **les trois premiers caractères** constituent l'**Orientation** à imposer aux **Rotors** avant de décoder un message. (Ou de le transmettre.) Du coup, lorsque l'on va initialiser la machine, seuls neuf lettres seront demandées puisque l'**Orientation** des **Rotors** aura déjà été saisie.

La feuille proposée en Fig.47 a été générée avec un petit utilitaire gratuit qui est disponible sur : <https://www.ciphermachinesandcryptology.com/en/codebook.htm>

Il ne fait que simuler des tables secrètes en ne générant que des codes au hasard et les affiche au

format de la page proposée. Avec les mêmes données initiales vous obtiendrez forcément des codes différents. Pour que nous ayons tous la même référence, je vous propose mon exemple dans le fichier **CODEBOOK-2-1942.txt**.



**ATTENTION** : Bien que ce soit un peu contradictoire, si vous voulez vous générer des pages, dans la zone rose ne pas prendre la référence **3-rotors M3-Kriegsmarine** qui contrairement à l'histoire propose trois **Rotors** parmi huit au lieu de cinq. Dans la zone verte il y a totale liberté pour proposer une date quelconque y compris future ...



## 12) Initialisation complète de l'ÉNIGMA virtuelle.

Outre la configuration des **Rotor** et le choix du **Réflexeur** on va devoir organiser le tableau des **FICHES croisées** et également traiter le **GROUPE d'identification**. L'objet de ce chapitre est de définir ces éléments et de les initialiser sur la codeuse. Avant de pouvoir manipuler pour tester ces éléments, il faut au préalable préparer la carte Arduino NANO.

### Nouveaux textes "définitifs" en EEPROM.

Opération maintenant routinière, on commence par inscrire ces dialogues dans la mémoire non volatile du microcontrôleur pour pouvoir ensuite les utiliser dans les prochains démonstrateurs. Dans ce but on téléverse **P00B\_Initialiser\_EEPROM.ino** prévu pour cette mission. Mis à part deux octets qui restent disponibles, l'EEPROM est entièrement saturée. On commence par activer le **Moniteur de l'IDE** et avec l'idéogramme  on initialise sa vitesse de transfert à **57600** baud. Puis on téléverse l'outil informatique **P00B** que l'on active ensuite en cliquant une deuxième fois sur . La fenêtre du moniteur se remplit et ressemble à la copie d'écran de la Fig.46 avec en particulier dans la zone verte les nouveaux textes inscrits. *(Seulement une partie d'entre eux a été modifiée pour conserver au maximum ceux qui étaient valides pour l'ancien menu car il faut reprendre les paramètres de tous les affichages qui changent. C'est un travail particulièrement indigeste)*

### Nouveau démonstrateur pour l'initialisation.

Nommé **P06\_Initialiser\_Entierement\_Enigma.ino** montre dans les informations en tête de listage que simplifier le MENU fait économiser 188 Octets de programme et surtout 26 Octets en mémoire dynamique ce qui est loin d'être négligeable. Dans **P05** il était déjà possible d'activer ou de suspendre le bruitage Morse avec la commande '**m**'. On se doute que le but consiste à rendre silencieux le petit module si on désire ne pas engendrer de gêne pour notre entourage. Le BIP sonore d'erreur qui se montre relativement bruyant est maintenant également suspendu simultanément avec le bruitage Morse. Notre machine sera alors totalement silencieuse. Par ailleurs, le clignotement de la LED bleue à la cadence du Morse n'est pas très utile. Aussi, le clignotement rapide pour indiquer à l'opérateur que le programme est en attente se fait sur cette dernière. *(Car avec le pion de RESET la LED Arduino en D13 n'est plus visible.)* Nouveau démonstrateur **P05** téléversé on peut passer à l'expérimentation de la fonction d'initialisation complète. Bien que le **GROUPE d'Identification** ne soit pas chiffré pour transmettre le message secret, il importe de le manipuler pour le début de chaque message. Pour éviter cette obligation, il fait partie de l'initialisation. C'est la commande '**g**' qui procèdera à la transmission automatique, nous n'aurons plus qu'à ajouter le **TEXTE** à coder.

### MANIPULATIONS :

- 01) Pour la forme "repartir à zéro" avec un RESET.
- 02) Frapper un '**m**' pour valider le bruiteur.
- 03) Frapper un '**i**' pour Initialiser la machine.
- 04) Frapper un '**o**' pour accepter. *(Confirmer évite d'entrer inopinément dans cette longue procédure.)*
- 05) Frapper et valider dans l'ordre '**4**' puis '**5**'. *(On est supposé être le 12/02/1942.)*
- 06) Le rotor de droite est le III mais on se trompe : Proposer '**8**'. Comme sur la machine il n'y en a que cinq potentiels on sursaute avec le BIP d'erreur et la machine redemande la valeur.
- 07) Dans la pièce voisine, Gertrude regarde un reportage sur son récepteur de télévision. Il ne faut pas la déranger. La commande '**m**' rendrait à nouveau notre petite boîte silencieuse, mais nous sommes en mode INITIALISATION et '**m**' n'aura pas cet effet. On recommence !
- 08) Faire un RESET qui par défaut nous livre une machine muette, le '**m**' est donc inutile. Vous pouvez toutefois l'utiliser deux fois pour constater son effet de type OUI/NON.
- 09) Frapper '**i**' suivi de '**o**' et saisir la chaîne "**453**" avant de valider. Le logiciel prévient qu'il n'a pris que le premier caractère, il passe au rotor suivant. *(Car il n'attend que pour le Rotor de gauche.)*
- 11) Nous faisons un effort d'attention et cette fois on propose les valeurs suivantes en validant à chacune d'elles : '**5**', '**3**', "**26**", "**23**", "**7**" et pour le **GROUPE d'Identification** on frappe les trois orientations initiales à indiquer : '**d**', '**I**' et '**p**'. *(I : lettre L minuscule.)*
- 12) Pour le réflecteur on impose '**b**'.

Maintenant on arrive à ce qui est nouveau, le programme nous demande de préciser la combinatoire du tableau des **Fiches croisées**. *(Zavez vu ? La LED bleue clignote rapidement !)*

- 13) Maintenant frapper **"bj"**, **"cq"**, **"di"**, **"ff"** là le logiciel n'accepte pas le cordon car il n'est pas croisé. Testons **"fmi"**, comme il y a plus de deux lettres la chaîne n'est pas acceptée et le logiciel reste sur la Fiche n°4. On tente un seul caractère avec **'f'**. Toujours sur la Fiche n°4 on propose la saisie de **"f="**. (*Décidément vous êtes vraiment distraits !*)
- 14) Expérimenter **"fb"**, comme le **'b'** a déjà été utilisé, sur la vraie machine on ne peut pas se tromper ainsi et ajouter une fiche en gigogne sur une autre déjà présente. Donc le logiciel reste sur cette fiche et se contente de signaler cette erreur de logique.

**REMARQUE :** Chaque fois qu'une fiche est enregistrée car correcte, la liste des lettres déjà utilisées est affichée. Ainsi, si par erreur on a un "doubleton", repérer la lettre en cause est bien plus simple.

- 15) Frapper **"fm"**, **"go"**, **"hr"**, **"kw"**, **"py"**, **"tu"** et enfin **"xz"**.

**NOTE :** Prendre garde au fait que les accentués sont acceptés et convertis en lettres simples. Par exemple **"çq"**, au lieu de **"cq"**, serait accepté et correctement enregistré.



À ce stade le logiciel nous demande de lui fournir les lettres relatives au **GROUPE d'Identification**. Toutefois dans le protocole de l'époque, les trois premières lettres correspondent aux orientations initiales des Rotors. Aussi, pour nous faciliter la saisie **le programme ne nous demande que les neuf dernières** lettres.

- 16) Tester **"iwq"**. Comme il n'y a pas assez de lettres le logiciel insiste.
- 17) Avec **"iwqgwyoà4"** punition identique car l'un des caractères n'est pas une lettre.
- 18) Expérimenter **"iwqgwyoadu"**. Le groupe n'est pas accepté car sa longueur dépasse 9 caractères.
- 19) Enfin proposer **"iwqgwyoad"**. Cette fois c'est la bonne. Il y a exactement neuf lettres et le **'à'** a été transposé en équivalent majuscule **'A'**, l'accentué est donc accepté.
- 20) Retrouvant le mode COMMANDES frapper **'c'** pour faire afficher la configuration actuelle de la

```

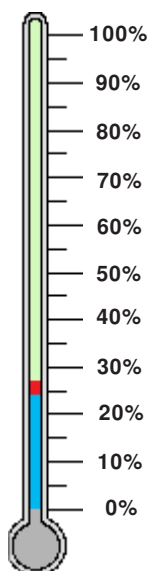
GROUPE d'identification -> [IWQGWYOAD]
COMMANDE -> [C]
>>> CONFIGURATION <<<
ROTOR de Gauche = IV Index de Gauche = 26 Orientation = D
ROTOR du Centre = V Index du Centre = 23 Orientation = L
ROTOR de Droite = III Index de Droite = 7 Orientation = P
REFLECTEUR = B
FICHE n°01 -> [B et J] FICHE n°02 -> [C et Q]
FICHE n°03 -> [D et I] FICHE n°04 -> [F et M]
FICHE n°05 -> [G et O] FICHE n°06 -> [H et R]
FICHE n°07 -> [K et W] FICHE n°08 -> [P et Y]
FICHE n°09 -> [T et U] FICHE n°10 -> [X et Z]
GROUPE d'identification = DLP IWQ GWY OAD
COMMANDE ->

```

Fig.49

codeuse virtuelle. On peut comparer les entités affichées à celles de la Fig.47 le formatage à l'écran étant celui de la Fig.49 dans laquelle les éléments clef sont mis en évidence par des couleurs. Dans les deux encadrés rouges on retrouve les lettres d'orientation initiales des **Rotors**. Dans la zone rose le **GROUPE d'Identification** est affiché au complet. Enfin dans la zone verte sont précisés les croisements effectués avec les fiches doubles.

Fig.50



➤ **À ce stade on refait un bilan sur l'évolution du programme.**

Avec la simplification du **MENU de base** et la mise en place de la procédure d'initialisation de la machine on a écrit un gros "morceau", car les erreurs de saisies potentielles sont légion et l'analyseur syntaxique doit effectuer un filtrage relativement fouillé. Chaque erreur détectée génère un texte spécifique gourmand en ressources, nous le savons. Sur le "thermomètre la zone bleue correspond à l'occupation de la mémoire réservée au programme lorsque l'on est passé au **MENU** optimisé de la Fig.45 soit environ 23%. La zone rouge correspond à l'implantation des deux nouvelles fonctions d'initialisation de la table des **Fiches croisées** et de la gestion du **GROUPE d'identification**. Il nous reste encore à mettre en place la fonction de correction des initialisations avec la commande **'e'**, ainsi que la sauvegarde et la restitution depuis l'EEPROM de ces données. La correction des erreurs risque encore de prendre pas mal de place car elle résulte d'un dialogue H/M qui nous le savons sera certainement glouton en Octets. Pas de panique, pour le moment on dispose encore d'une place confortable et surtout les nouveaux dialogues seront directement implantés dans le programme. C'est un avantage, car il n'y aura plus au préalable à les logger en EEPROM avec l'outil spécifique **P00B**. Tout va bien pour le moment et l'avenir est tout tracé ...

### 13) Pouvoir facilement éditer l'Initialisation de la machine.

L'exercice abordé dans le chapitre précédent montre que pour initialiser entièrement l'Enigma virtuelle, c'est pas moins de 39 entités alphabétiques ou numériques qu'il faut saisir tout en respectant avec rigueur un document tels que celui de la Fig.47 présenté en page 29. C'est tellement facile de se tromper de ligne, voir de commettre une erreur de frappe qui passe inaperçue dans l'analyseur syntaxique, que de ne pas avoir à reprendre l'intégralité du protocole pour corriger s'avère indispensable à la qualité opérationnelle de notre programme d'exploitation. C'est dans cette optique que la commande 'e' a été insérée dans le **MENU de base** dès le début des études. C'est avec le démonstrateur **P07\_Editeur\_de\_correction\_de\_saisie.ino** que l'on va développer cette facette du logiciel. Naturellement on va "se contenter de reprendre" **P06** et y ajouter cette nouvelle fonction.

#### > L'Editeur de la configuration d'Enigma.

Comme le démonstrateur **P07** est bien plus complet que le simple ajout de la commande 'e', il suppose qu'une INITIALISATION d'Enigma est disponible en EEPROM. Sous cette hypothèse il charge automatiquement cette dernière et après avoir affiché le **MENU de base** il liste l'état actuel de configuration de la machine. Comme pour le moment vous n'avez encore jamais sauvegardé une telle configuration, jusqu'à le faire il faudra ne pas s'étonner d'un affichage dont les valeurs sont nulles ou non affichées. Tout va rentrer dans l'ordre durant ce chapitre. Commencer par téléverser **P07** et activer le **Moniteur de l'IDE**. On peut éditer la CONFIGURATION par "secteur" et dans un ordre quelconque. Par secteur il faut comprendre : [**Rotors** et **Réflexteur**], [**Fiches croisées**] ou [**GROUPE d'identification**]. Passons à l'expérimentation.

#### MANIPULATIONS :

- 01) Frapper "eg" et valider. (*On est toujours le 12/02/1942.*)
- 02) Entrer "iwqgfyoad" et valider. Dans cet exercice c'est volontairement que l'on s'est "trompé" et que le 'f' a été saisi à la place du 'w'. (*On ne donne pas les trois premières lettres.*)



**NOTE :** Bien qu'à tout moment la commande 'c' permet d'afficher *l'état initial* de la codeuse, en sortie d'éditeur ce dernier *est automatiquement présenté* pour que l'opérateur puisse vérifier le résultat de son intervention.

- 03) Proposer "ef" pour corriger le tableau des **Fiches croisées**.
- 04) C'est reparti pour "bj", "çq", "di", "fm", "go", "hr", "kw", "py", "tu", "xz". (*La lettre 'ç' est acceptée et translatée en équivalent majuscule sans la cédille.*)
- 05) Enfin on édite le **Brouilleur** avec "er" pour commencer par les **Rotors**.
- 06) Dans l'ordre on propose '4', '5', '3' puis "26", "23", "7" et terminer par 'd', 'l', 'p'. Pour mémoire 'l' est un 'L' minuscule dans cette série de lettres.
- 07) On complète avec 'b' pour indiquer le **Réflexteur**.

Fig.51


DLP	IWQ	GWY	OAD
DLP	IWQ	GFY	OAD

On notera au passage que le **GROUPE d'identification** est

entièrement initialisé, les trois premiers caractères correspondant à l'orientation initiale ont été recopiés en tête du **GROUPE d'identification**. On a "dans le désordre" corrigé entièrement la configuration initiale, vérifions une dernière fois la cohérence de l'ensemble avec la feuille de cryptage.

**- Mazette, on s'est trompé dans le groupe d'identification !** (*Voir la Fig51*)

C'est dans ce genre de situation que l'on va apprécier l'éditeur de Configuration, car il n'y a pas à tout reprendre. On va se contenter de corriger uniquement le **GROUPE d'identification**.

- 08) Imposer "eg" puis "iwqgwyoad" pour réintroduire cette donnée.
- 09) C'est fait, la configuration de notre Enigma virtuelle correspond exactement à celle qui dans la feuille de cryptage occupe la ligne du 12/02/1942. Comme elle est cohérente et que l'on va certainement s'en servir par la suite, on frappe 's' et on confirme par 'o'. (*Il y a demande de confirmation car il ne faudrait pas perdre un enregistrement en ayant par mégarde frappé la lettre 's' durant le mode COMMANDES.*)
- 11) Le programme est assez laconique, en principe c'est fait mais il ne le signale pas. Aussi, pour vérifier c'est facile : Effectuez un RESEET tout en ouvrant le **Moniteur**. Il suffit dans la fenêtre de l'**IDE** de cliquer sur . Chic chic chic la configuration rechargée est correcte.



**A**utant en lecture une mémoire EEPROM est inusable, autant en écriture elle présente une limite de fiabilité. Les fournisseurs de ce type de composants précisent qu'une EEPROM n'est garantie et fiable que pour 100.000 écritures. Ensuite elle va commencer à présenter aléatoirement des mauvaises inscriptions. Évidemment ce n'est qu'un ordre de grandeur.

**CONCLUSION :** *ne jamais utiliser l'EEPROM comme de la RAM dans laquelle on passe son temps durant le déroulement d'un programme à y écrire des variables qui changent rapidement de valeurs, la mémoire non volatile arriverait rapidement à sa destruction.*

*Quand on stocke des valeurs qui ne doivent pas être perdues si se présente une coupure secteur, il importe pour le programmeur de s'assurer de la durée de vie du composant*, ce que nous allons faire immédiatement. Les textes figés en EEPROM ne seront modifiés que durant le développement du programme, donc très peu de réécriture seront effectuées. Supposons que l'on sauvegarde avec 's' dix fois par jours, tous les jours. Avant que l'EEPROM ne soit usée il va s'écouler  $100000 / 10 = 10000$  jours soit 27 années complètes. Je doute fort que ce petit appareil ne soit utilisé de cette façon. Il y aura longtemps que le conflit 39/45 sera terminé et que notre Uboat aura rejoint un musée ! Aussi, coté "usure de l'EEPROM" nous n'avons rien à craindre.

### ➤ Restitution d'une sauvegarde EEPROM.

**R**echarger automatiquement le contexte lorsque l'on redémarre le programme d'exploitation sera généralement suffisant à l'usage. Il est peu vraisemblable que l'opérateur puisse par erreur fausser l'initialisation rechargée lors du RESET. Comme ce n'est jamais totalement exclus et pour éviter d'avoir à effectuer un RESET, on peut recharger à tout moment les données avec la commande 'r'. Le mieux consiste encore à expérimenter cette possibilité :

#### MANIPULATIONS : (SUITE)

- 12) Faire un RESET pour repartir sur une base commune.
- 13) Frapper "eg" et valider.
- 14) Entrer "aaaccbb" et valider pour perdre le GROUPE d'identification.
- 15) On change d'idée pour revenir à la date du 12/02/1942. On frappe 'r' et on confirme par 'o'. (Il y a demande de confirmation car il ne faudrait pas perdre une configuration en cours de modification en ayant par mégarde frappé la lettre 'r' durant le mode COMMANDES.)

OUF on a retrouvé nos données qui resteront en EEPROM jusqu'à la prochaine commande 's'. Dans le MENU de base la commande 'g' a été enlevée au profit d'une transmission automatique du GROUPE d'identification. Testons ce comportement intégré dans P07 :

- 16) Proposer '&' Puis quelques lettres en validant à chaque fois pour tester le cryptage individuel.
- 17) Retour au mode COMMANDES avec '&' suivi de 't' pour le mode cryptage "normal".
- 18) Passage au chiffage avec '&' puis le début du message secret "Bonjour les amis.". On constate que le préambule est listé, puis à la ligne suivante commence le message crypté. (Ici les caractères sont simplement décalés d'une lettre. Les espaces et le point frappés par habitude sont ignorés.)
- 19) Continuer le message secret avec "Tout va bien il n'y a pas de problème.". L'apostrophe n'est pas acceptée seuls les caractères qui précèdent sont transcodés.
- 20) On poursuit avec "yapasdeproblème. Fin de transmission.". Ne comportant que des caractères valides le texte est crypté et "ajouté" au listage à l'écran.
- 21) Frapper '&' pour revenir au mode COMMANDE.

 *Le message est supposé entièrement envoyé.* Donc le prochain chiffage sera supposé pour un autre message. Du coup le programme réitère l'affichage du GROUPE d'identification.

- 22) Frapper '&' pour coder un nouveau message puis "aaaaaaaaaazzzzzzzzzzeeeeeeeee".
- 23) Continuer avec "rrrrrrrrrrtttttttttyyyyyyyyy". (Dix lettres identiques à chaque fois.)
- 24) Finir avec "ooooooooopppppppppppqqqqqqqqqssssssssssddddd".


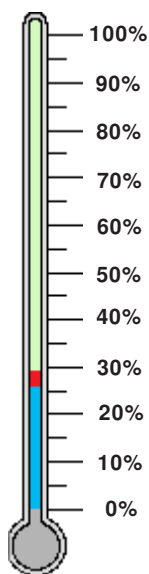
 Si on ne propose pas de caractères interdits ou si on reste dans des phrases de moins de 61 caractères le formatage par paquets de six lettres est "continu".

Fig.52



A ce stade du développement on dispose de pratiquement tous les éléments fonctionnels virtuels de la machine mis à part le mécanisme qui fait tourner les Rotors. On va pouvoir commencer à câbler virtuellement cette Énigme immatérielle. Avant de commencer cette nouvelle phase dans notre projet, un bilan me semble salutaire. Avant de modifier **P07** directement issu de **P06**, le programme occupait 26% de la place réservée au programme. En ayant ajouté l'Editeur de configuration j'étais pessimiste en affirmant que ce dernier risquait de se montrer boulimique. Au final il ne "gloutonne" que 214 octets soit à peine 1% de l'espace réservé. Surtout, la mémoire dynamique utilisée n'augmente que de huit Octets. C'est assez normal, car les nouveaux textes du dialogue ne font que remplacer certains qui étaient déjà dans le "Sketch". En ajoutant les fonctions EEPROM, l'automatisation de la transmission en mode TEXTE du **GROUPE d'identification** pour toutes ces fonctions on "consomme à peine" 664 Octets. Nous avons encombré moins du tiers de l'espace réservé au programme, autant dire qu'il en reste à mon avis largement plus qu'il n'en faut pour émuler la machine. Donc, ce jour du "12/02/1942" je suis serein et l'on va enfin pouvoir passer au développement de CRYPTAGE proprement dit. On va y aller progressivement en commençant par les fonctions qui me semblent les moins compliquées pour terminer par les plus critiques.

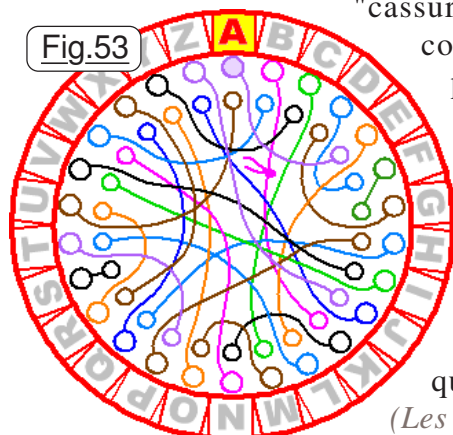
#### 14) Un changement important de stratégie.

Historiquement j'avais à ce stade envisagé de rédiger le chapitre "*Émuler l'Indexation interne des Rotors*" pour commencer à agencer le comportement virtuel des organes de la machine. Dans cette optique j'ai commencé à élaborer le démonstrateur **P8** accompagnant ledit chapitre. C'est alors que j'ai constaté une grosse faille dans la façon de coder en EEPROM les **Rotors**. Ces composants de notre magasin virtuels sont complètement à reprendre ce qui va nous obliger à corriger le contenu du haut de la mémoire non volatile. C'est la vie informatique ...

L'idée initiale qui était séduisante consistait à représenter un **Rotor** par un tableau Fig.33 dans lequel le codage des cellules correspondait à la transposée du caractère présenté en entrée du **Rotor** considéré. Il se trouve que cette approche s'est avérée inexploitable lorsque les premières séquences de **P8** ont été rédigées. Cette "catastrophe" résulte du fait que l'on n'avait pas encore envisagé de faire tourner cet élément virtuellement. Hors, dès qu'il change d'orientation, le codage est complètement chamboulé et le tableau représentatif devient inutilisable. L'intégralité de ce chapitre va concerner le **Rotor I**, les autres seront traités de façon analogue. *La nouvelle approche consiste à coder pour ce composant immatériel son câblage interne. Puis, en fonction des manipulations virtuelle effectuée, c'est ce câblage fictif qui subira les rotations et configurations envisagées.* Pour comprendre la nouvelle approche, on va y aller très progressivement.

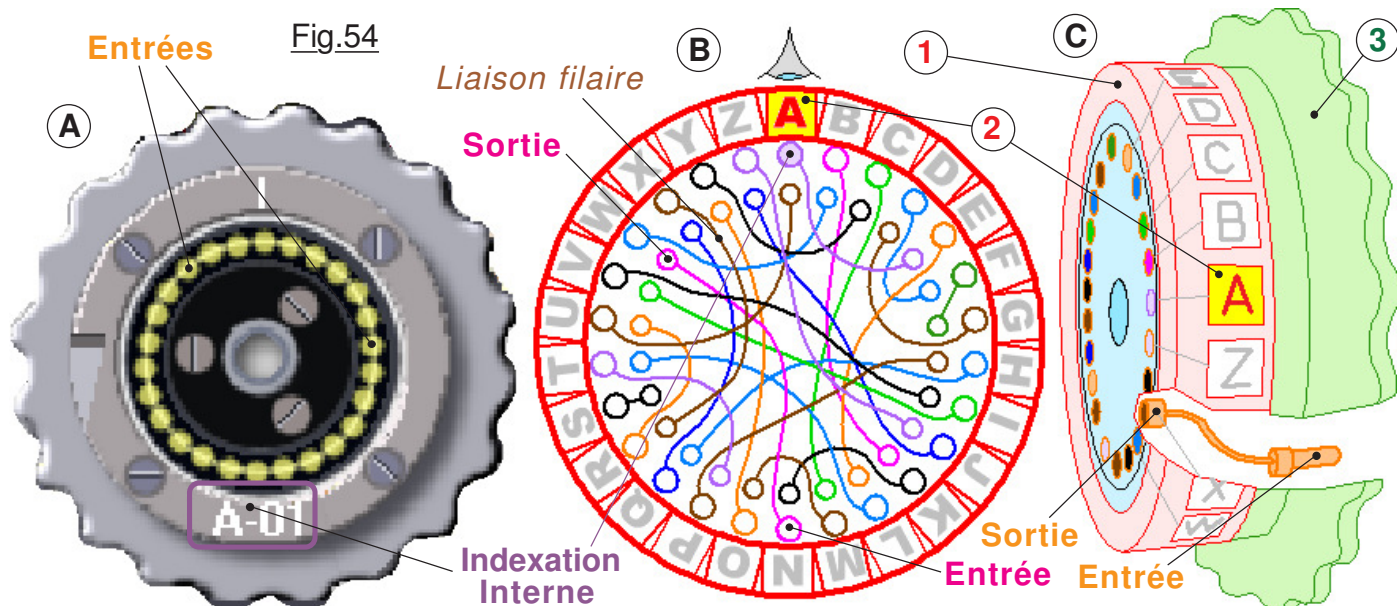
#### ➤ **La représentation graphique du Rotor I.** (Fig.53)

Durant les chapitres qui vont suivre, nous allons utiliser une représentation graphique des **Rotors** qu'il importe de bien assimiler si l'on veut comprendre les explications et les décisions qui s'en dégagent. Considérons la Fig.54 sur laquelle est représenté en **A** le **Rotor** numéro **I** tel qu'il se présente à l'opérateur face "**Entrées**" avec ses contacts électriques mâles télescopiques à ressorts. En **C** le même composant est montré en perspective, c'est l'autre face coté **Sorties** avec dans la "cassure" la représentation interne d'une *Liaison filaire* entre les deux contacts. Enfin en **B** la représentation du Rotor telle qu'elle sera proposée dans ce chapitre, c'est la face **Entrées** qui est dirigée vers nous. On retrouve en **1** la **Bague** qui peut être **Indexée** en **Interne** sur le corps **3** de l'élément. En **2** se trouve la lettre orientée visible de la fenêtre de la machine lorsque le **Rotor** à tourné suite à la frappe d'un caractère sur son clavier. Pour faciliter l'observation de cet organe du **Brouilleur** les câblages internes sont mis en évidence par des couleurs différentes. Il importe de noter que le contact électrique du fil interne n°1 est le seul dont le "gros cercle" qui représente l'**Entrée** est également colorié à l'intérieur en violet.



(Les "petits" cercles représentent les sorties du composant.)

Dans cet exemple l'**Indexation interne** est la n°01 soit en lettre **A** comme dans l'encadré violet.



### ► Le codage des Rotors en EEPROM.

Représentation en EEPROM ou dans le programme d'exploitation : même combat. C'est à dire que dans notre magasin virtuel **les cinq rotors sont tous stockés avec un Indexage Interne sur la position n°1**. Dans le programme d'exploitation d'Enigma on va prévoir six tableaux de type **byte** qui représenteront Les trois **Rotors** avec traversée dans le sens **Direct** et dans le sens **Réfléchi**. À l'initialisation ces six tableaux seront rechargés à partir de l'EEPROM. Ils seront représentatifs de la configuration du moment dans le **Brouilleur**. Chaque fois que l'on débutera un cryptage, on va devoir dans l'ordre effectuer virtuellement les manipulations de l'opérateur fictif suivantes :

- Prendre le **Rotor** de **Gauche**. (*Le recopier depuis l'EEPROM dans son tableau représentatif.*)
- L'**Indexer en Interne** pour le sens **Direct** et le sens **Réfléchi**.
- L'**Orienter** sous la fenêtre de la machine pour le sens **Direct** et le sens **Réfléchi**.
- Prendre le **Rotor** du **Centre**. (*Le recopier depuis l'EEPROM dans son tableau représentatif.*)
- L'**Indexer en Interne** pour le sens **Direct** et le sens **Réfléchi**.
- L'**Orienter** sous la fenêtre de la machine pour le sens **Direct** et le sens **Réfléchi**.
- Prendre le **Rotor** de **Droite**. (*Le recopier depuis l'EEPROM dans son tableau représentatif.*)
- L'**Indexer en Interne** pour le sens **Direct** et le sens **Réfléchi**.
- L'**Orienter** sous la fenêtre de la machine pour le sens **Direct** et le sens **Réfléchi**.

Dans ce chapitre et ceux qui suivent, on ne va prendre en compte que le **Rotor n°1**.

Fig.55

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
E	K	M	F	L	G	D	Q	V	Z	N	T	O	W	Y	H	X	U	S	P	A	I	B	R	C	J

### Codage de I en EEPROM sens Direct. Bague en 1.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
4	9	10	2	7	1	23	9	13	16	3	8	2	9	10	18	7	3	0	22	6	13	5	20	4	10

Considérons le haut du tableau de la Fig.55 qui montre en rouge les lettres qui "entrent" dans le **Rotor** et en bleu celles qui sont permutées en interne vers la sortie. Pour représenter le câblage, on va comme présenté dans l'encadré jaune symboliser **à combien de contacts en interne se trouve plus loin le plot électrique de sortie**. Par exemple si l'on prend en exemple le fil vert clair actuellement en **C**, il ressort en **M** dix contacts plus loin. Dans le tableau du sens **Direct** cette cellule d'indice 2 sera codée **10**. Autre exemple : La cellule d'indice 11, le fil bleu clair actuellement en **L**, il ressort en **T** huit contacts plus loin. Les cinq rotors seront représentés de cette façon.





Bande de Dudulettes et de Dudules, vous n'avez même pas remarqué que sur la Fig.53 le fil rose est épaissi, et que sur la Fig.54 il a été changé !

## 15) Émuler l'Indexation interne des Rotors.

Nous n'avons pas le choix, car pour utiliser le démonstrateur **P8** il faut au préalable avoir les bons **Rotors** en magasin. Donc on doit commencer par corriger les données en EEPROM avec **P00C\_Initialiser\_EEPROM.ino** qui ne réinscrira que la zone concernée. (*Téléverser et activer ce programme en premier ... La routine !*) On corrige ainsi l'ensemble des cinq éléments de la machine avec le nouveau codage. Pour conserver les facilités apportées par le dialogue Homme/Machine on va copier **P7** pour partir avec le code actuel dans **P08\_Rotation\_du\_Rotor\_de\_Droite.ino** qui intègre comme on va l'expérimenter également l'Orientation Initiale et la Rotation sous la fenêtre de la codeuse. **Notons au passage deux petites améliorations par rapport à P7 :**

- Changement de stratégie pour le bruiteur : La commande '**m**' n'affecte maintenant que le Morse manipulé avec confirmation sonore. Pour suspendre ou valider le BIP d'erreur la commande '**b**' a été introduite. L'affichage du **Menu de base** intègre cette nouvelle commande.
- Sur caractère invalide en cryptage sous mode TEXTE l'affichage de l'erreur est entièrement revu.

### ➤ Représentation du Collecteur en liaison avec le Rotor de Droite.

Autre organe de la crypteuse qu'il va falloir émuler, le **Collecteur** représenté en **B** sur la Fig.56 qui constitue le maillon le plus à droite du **Brouilleur** et effectue la liaison électrique entre le **Rotor** de **Droite** en **A** avec le **Clavier** et le tableau des **FICHES croisées**. Le dessin de la Fig.56 est indissociable de celui de la Fig.57 qui contient maintenant les lettres minuscules qui figurent les contacts entre le **Collecteur** et le **Rotor**. Sur la Fig.57 on suppose que l'on observe le **Rotor** dans le sens de la flèche jaune et que le **Collecteur** est transparent. Il est important de bien comprendre que le **Collecteur** est immobile sur la machine, donc quoi qu'il arrive le repérage des contacts électriques en lettres minuscules sera immuable. Ce sont uniquement la **Bague extérieure** et le câblage interne du **Rotor** qui vont changer d'orientation en utilisation de la machine. Chaque fois que l'on installera le **Rotor** n°1 sur la codeuse avec l'**Indexation** sur la référence n°1 on obtiendra la structure

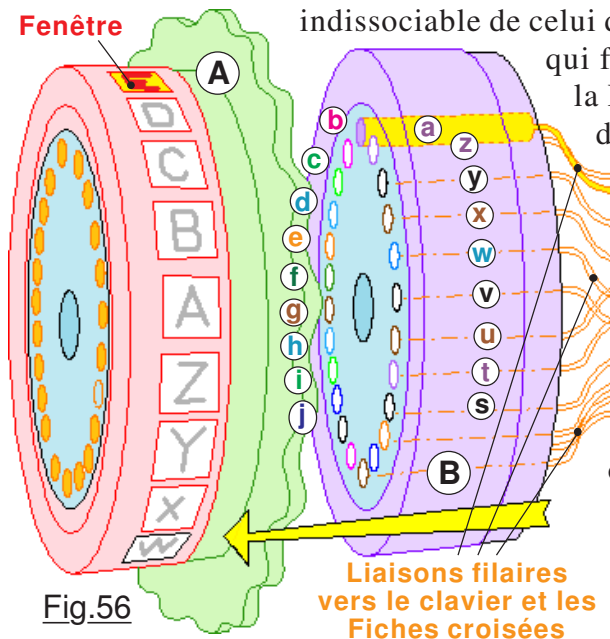


Fig.56

de la fig.57, c'est à dire la configuration de l'élément pris dans le magasin (*Recopié depuis l'EEPROM.*) avant d'avoir été placé sur la machine car il faut au préalable imposer l'**Indexation** interne, puis sur le **Brouilleur** respecter sa Position et enfin son Orientation. En conclusion, sur un dessin tel que celui de la Fig.57 on voit la **Bague extérieure** avec les lettres majuscules qui se trouve au fond, (*En jaune l'orientation sous la fenêtre.*) le câblage interne dans le Rotor, et enfin le Collecteur fixe vu par transparence dont les contacts électriques en entrée de **Rotor** sont repérés par les lettres minuscules.

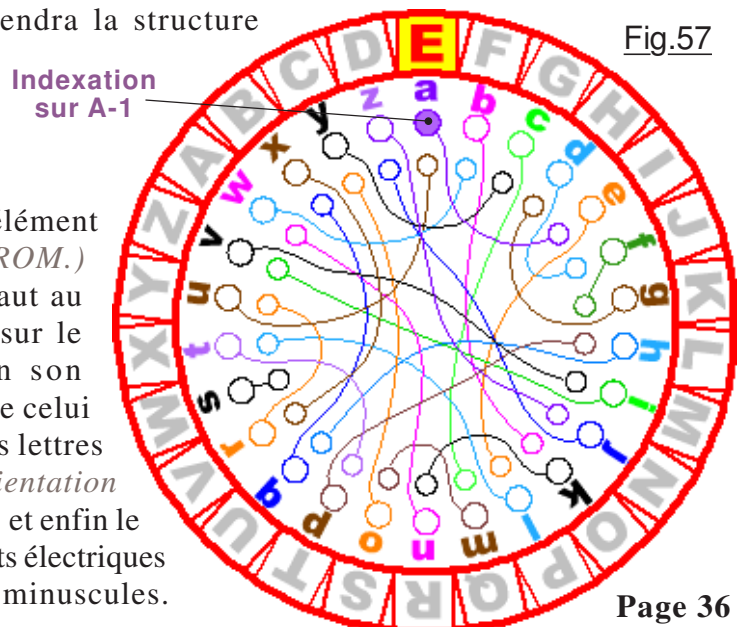


Fig.57

### ➤ Indexation du Rotor de Droite dans le sens Direct.

Lorsque l'opérateur saisit le Rotor n°1 dans sa main, dans le coffret il est rangé "par discipline" avec une Indexation interne en référence A-1. C'est la configuration de la Fig.58 en A. Supposons maintenant que l'opérateur tourne la Baguette par rapport au corps du composant et indexe la référence B-2. Quand il place le Rotor sur la codeuse avec sa lettre A vers la fenêtre, en observation "relative" nous obtenons la configuration globale de la Fig.58 en B. Pour traduire cette organisation, le tableau représentatif du sens Direct dans le programme sera celui résumé en bas du dessin. Au départ il y a correspondance directe entre la Baguette vue par la fenêtre en lettres majuscules et les Entrées en lettres minuscules. Le câblage interne en relatif à tourné dans le sens horaire d'une position. Pour traduire informatiquement ce nouveau contexte il suffit dans le tableau

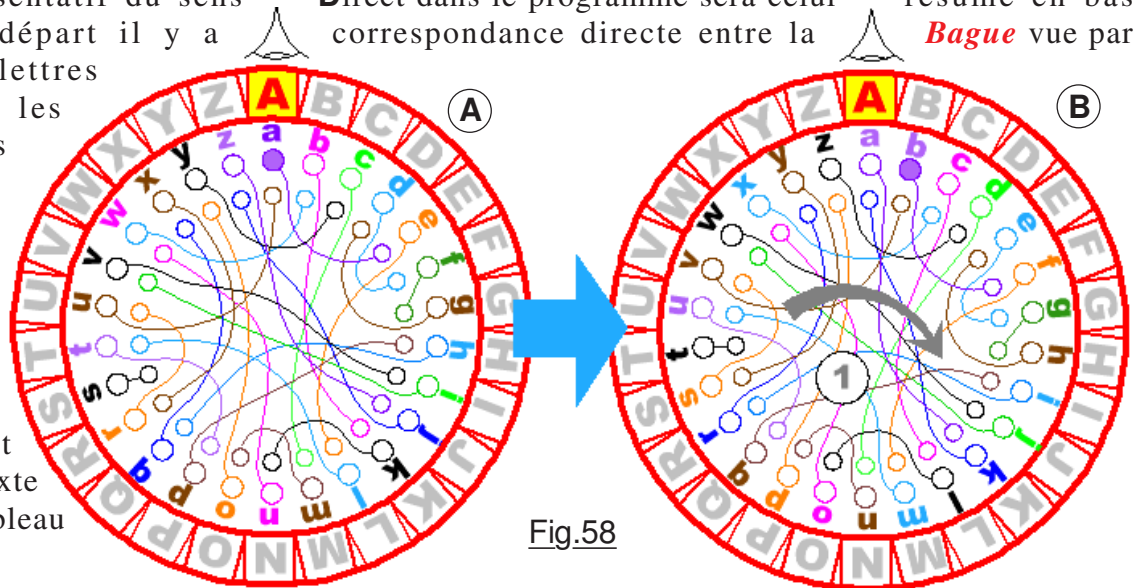


Fig.58

<b>Fenetre</b>	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
<b>Entrée</b>	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
<b>1</b>	10	4	9	10	2	7	1	23	9	13	16	3	8	2	9	10	18	7	3	0	22	6	13	5	20	4
<b>Sortie</b>	k	f	l	n	g	m	h	e	r	w	a	o	u	p	x	z	i	y	v	t	q	b	j	c	s	d

de la Fig.55 de décaler à droite d'une position le codage des écarts entre les contacts d'une position. Informatiquement la technique consiste à sauvegarder la cellule de droite, puis d'effectuer un "Shift" des 25 cellules de gauche et de restituer la valeur sauvegardée dans l'emplacement de gauche, c'est à dire la cellule d'indice zéro. Cette suite d'opérations représentée sur la Fig.59 concrétise en fin de compte ce que l'on nomme en informatique un opérateur Rotation Logique à Droite. Fig.59



### ➤ Utilisation de P8 pour tester l'Indexation de la position n°2 dans le Rotor.

Actuellement le programme P08\_Rotation\_du\_Rotor\_de\_Droite.ino est paramétré pour ne faire qu'installer le Rotor n°1 à Droite sur la machine, Indexé en B-2. Pour cette première manipulation il ne tourne pas et l'on se contente de proposer toutes les lettres en entrée pour constater les transpositions données dans le tableau de la Fig 58 en ligne du bas. On suppose ici que vous avez téléversé et activé P00C, puis P08 et que le Moniteur vient d'afficher le Menu de base. En préambule on va changer complètement l'initialisation de la machine pour obtenir celle de la Fig.60 donnée en page 38. (Noter que par défaut le BIP sonore d'erreur est validé.)

#### MANIPULATIONS :

- 01) Débuter l'expérimentation par avec "er" pour modifier la configuration des Rotor.
- 02) Frapper '3' puis '2' et '1' et valider à chaque fois.
- 03) Proposer ensuite '1' puis '1' et '2' et valider à chaque fois pour les indexations.
- 04) Puis indiquer 'a', 'a', 'a' pour les orientations initiales sous la fenêtre.
- 05) Enfin imposer le Réflecteur avec 'b', le logiciel affiche la nouvelle configuration.

- 06) Comme on va utiliser cette initialisation plusieurs fois, on la sauvegarde en EEPROM avec 's' confirmé par 'o'.
- 07) Proposer l'erreur volontaire 'a'.
- 08) Frapper un 'b' pour passer en "sourdine" puis recommencer avec 'a' pour vérifier le mode silencieux.
- 09) Si vous préférez la présence du BIP sonore sur erreur, réitérer 'b'.
- 10) Passer en mode CRYPTAGE avec '&'.
- 11) Donner toutes les lettres de l'alphabet dans l'ordre en validant à chaque fois 'a', 'b', 'c', 'd' ... 'x', 'y', 'z' et 'a', et vérifier que leur transposition est bien celle de la ligne du bas du tableau de la Fig.58 en bleu. On constate au passage que pour la rotation sous la fenêtre de la machine il y a bien rotation, mais pas pour le circuit interne du rotor proprement dit.

```
>>> CONFIGURATION <<<
ROTOR de Gauche = III Index de Gauche = 1 Orientation = A
ROTOR du Centre = II Index du Centre = 1 Orientation = A
ROTOR de Droite = I Index de Droite = 2 Orientation = A
REFLECTEUR = B
FICHE n°01 -> [B et J] FICHE n°02 -> [C et Q]
FICHE n°03 -> [D et I] FICHE n°04 -> [F et M]
FICHE n°05 -> [G et O] FICHE n°06 -> [H et R]
FICHE n°07 -> [K et W] FICHE n°08 -> [P et Y]
FICHE n°09 -> [T et U] FICHE n°10 -> [X et Z]
GROUPE d'identification = AAA IWQ GWY OAD
```

Fig.60

### > Indexation du Rotor de Droite dans le sens Réfléchi.

Avec cette expérience, on va encore rester en **Indexation B-2** mais cette fois on va vérifier la transposition en **Réfléchi** dans le **Rotor I**. Comme ces manipulations se font encore avec le démonstrateur **P8**, on se doute qu'il faut le modifier. Pour vous rendre cette opération élémentaire, nous allons nous contenter de modifier un jeu de paramètres situés en tête du programme. Dans ce

Fig.61

```
#define Direct true
#define Reflechi false
#define Indexer_en_Direct true
#define Indexer_en_Reflechi false
#define Rotation_en_Direct false
#define Rotation_en_Reflechi false
```

Fig.62

```
#define Direct false
#define Reflechi true
#define Indexer_en_Direct false
#define Indexer_en_Reflechi true
#define Rotation_en_Direct false
#define Rotation_en_Reflechi false
```

but, on trouve en haut du listage les six déclarations de la Fig.61 toutes relatives à des paramètres booléens. Pour Passer en mode **Réfléchi** avec une **Indexation B-2** il suffit de modifier ces valeurs comme montré sur la Fig.62 les rotations restant à **false**. On aboutit à la configuration de la Fig.63 qui est strictement identique au dessin de la Fig.58 **B** puisqu'il s'agit en fait de la même configuration. Notez au passage que sur le croquis les lettres minuscules adoptent la couleur du "gros cercle extérieur", donc celle de la ligne filaire intérieure. Par contre, dans le tableau de la Fig.64 représentatif de cette situation, les lettres représentant les cheminements en retour adoptent les couleurs des "petits cercles intérieurs" le mouvement du courant électrique se faisant "en sens inverse". Ce détail est mis en évidence par les petites flèches sur certains des fils électriques virtuels.

Fig.63

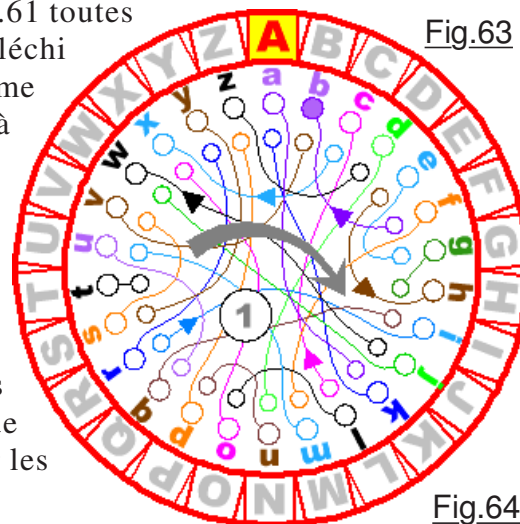


Fig.64

<b>Fenetre</b>	<b>A</b>	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
<b>Entrée</b>	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
<b>1</b> →	10	20	21	22	3	22	24	25	8	13	16	17	19	16	23	24	4	17	6	0	18	23	13	17	19	16
<b>Sortie</b>	k	v	x	z	h	b	e	g	q	w	a	c	f	d	l	n	u	i	y	t	m	s	j	o	r	p

### MANIPULATIONS :

- 01) Modifier comme montré en Fig.62 le démonstrateur **P8** et le téléverser.
- 02) Activer le **Moniteur** et consigner un '&' pour passer en mode CRYPTAGE.
- 03) Puis frapper 'a', 'b', 'c' ... 'x', 'y', 'z' toutes les lettres dans l'ordre, il est ainsi plus facile de comparer avec les caractères de la ligne du bas de la Fig.64 indiquant les transposés en sortie.



**NOTE :** Pour simplifier l'écriture des Manipulations il ne sera plus précisé s'il faut valider après chaque caractère ou s'il convient de frapper toute une chaîne avant d'entériner. On se contentera d'observer la rédaction qui reprend la syntaxe du c++. Si c'est un caractère individuel il est encadré par deux apostrophes comme '&' par exemple. Pour une chaîne on encadre avec des guillemets ce qui peut donner "abc3def".

**REMARQUE :** Quand on observe la ligne des nombres bleus traduisant l'écart entre les contacts situés aux deux extrémités d'une ligne filaire, pour indexer une position voisine il suffit aussi bien en **Direct** qu'en **Réfléchi** d'effectuer une **Rotation Logique à Droite** d'un BIT pour réaliser une **Indexation**. On se doute que si l'on veut Indexer **H-8** par exemple il faudra 7 rotations.

➤ **Indexation du Rotor de Droite en M-13 dans le sens Direct.**

Dans cet exemple il serait possible de choisir n'importe quelle **Indexation** entre 1 et 26. La valeur de 13 est choisie car elle se trouve au milieu. Rien n'interdit de tester d'autres **Indexations**, il suffit de reprendre l'exercice qui va suivre ... et de refaire le dessin de la Fig.65 en faisant tourner le centre de la quantité idoine. Par manipulations, si l'**Indexation** désirée est la n°13, le programme **P8** va automatiquement effectuer 12 **Rotations Logiques à Droite** d'un BIT pour configurer le **Rotor** considéré. On va alors obtenir le tableau de la Fig.66 que l'on peut comparer au dessin de la Fig.65 les couleurs des lettres minuscules étant à nouveau celles des "gros cercles extérieurs". Dans **P8** il faut revenir à la combinatoire logique de la Fig.61 car on retrouve le sens **Direct** pour le cryptage et pour l'**Indexation**.

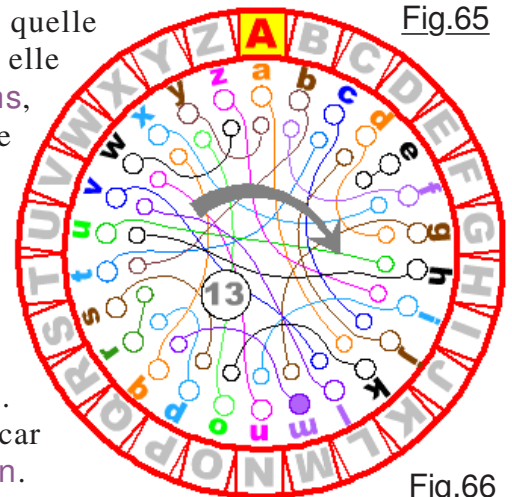


Fig.65

<b>Fenetre</b>	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>	<b>F</b>	<b>G</b>	<b>H</b>	<b>I</b>	<b>J</b>	<b>K</b>	<b>L</b>	<b>M</b>	<b>N</b>	<b>O</b>	<b>P</b>	<b>Q</b>	<b>R</b>	<b>S</b>	<b>T</b>	<b>U</b>	<b>V</b>	<b>W</b>	<b>X</b>	<b>Y</b>	<b>Z</b>
<b>Entrée</b>	<b>a</b>	<b>b</b>	<b>c</b>	<b>d</b>	<b>e</b>	<b>f</b>	<b>g</b>	<b>h</b>	<b>i</b>	<b>j</b>	<b>k</b>	<b>l</b>	<b>m</b>	<b>n</b>	<b>o</b>	<b>p</b>	<b>q</b>	<b>r</b>	<b>s</b>	<b>t</b>	<b>u</b>	<b>v</b>	<b>w</b>	<b>x</b>	<b>y</b>	<b>z</b>
<b>12 ➡</b>	<b>10</b>	<b>18</b>	<b>7</b>	<b>3</b>	<b>0</b>	<b>22</b>	<b>6</b>	<b>13</b>	<b>5</b>	<b>20</b>	<b>4</b>	<b>10</b>	<b>4</b>	<b>9</b>	<b>10</b>	<b>2</b>	<b>7</b>	<b>1</b>	<b>23</b>	<b>9</b>	<b>13</b>	<b>16</b>	<b>3</b>	<b>8</b>	<b>2</b>	<b>9</b>
<b>Sortie</b>	<b>k</b>	<b>t</b>	<b>j</b>	<b>g</b>	<b>e</b>	<b>b</b>	<b>m</b>	<b>u</b>	<b>n</b>	<b>d</b>	<b>o</b>	<b>v</b>	<b>q</b>	<b>w</b>	<b>y</b>	<b>r</b>	<b>x</b>	<b>s</b>	<b>p</b>	<b>c</b>	<b>h</b>	<b>l</b>	<b>z</b>	<b>f</b>	<b>a</b>	<b>i</b>

Fig.66

## MANIPULATIONS :

- 1) Modifier comme montré en Fig.61 le démonstrateur **P8** et le téléverser.
- 2) Activer le **Moniteur** et proposer "er" pour modifier l'indexation.
- 3) Frapper dans l'ordre '3', '2', '1', '1', '1', '13', 'a', 'a', 'a' et 'b' pour le **Réflécteur**.
- 4) Proposer un '&' pour CRYPTER.
- 5) Puis frapper 'a', 'b', 'c' ... 'x', 'y', 'z' toutes les lettres dans l'ordre, il est ainsi plus facile de comparer avec les caractères de la ligne du bas de la Fig.66 indiquant les transposés en sortie.

## 16) Aller / Retour dans le Rotor de Droite sans Indexation.

Dans la mesure où il n'y a aucune modification apportée par la l'**Indexation** ou la **Rotation** et que le **Réflécteur** retourne directement le même Caractère, en **Sortie** on doit retrouver la lettre donnée en **Entrée** du **Brouilleur**. C'est un cas particulier où la **Sortie** revient directement sur l'**Entrée** ce qui ne se produira jamais dans la pratique car le **Réflécteur** réalise systématiquement des permutations entre arrivée et retour du signal électrique. C'est du reste une faiblesses d'Enigma car Turing qui en a tenu compte en 1941 pour casser le code de la codeuse savait qu'une lettre ne pouvait jamais être cryptée en elle même. Dans notre cas, c'est un moyen fictif de vérifier le câblage interne virtuel des **Rotor**.

```
#define Direct true
#define Reflechi true
#define Indexer_en_Direct false
#define Indexer_en_Reflechi false
#define Rotation_en_Direct false
#define Rotation_en_Reflechi false
```

Fig.67

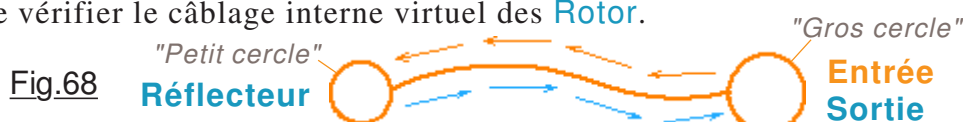


Fig.68

**MANIPULATIONS :**

- 01) Modifier comme montré en Fig.67 le démonstrateur **P8** et le téléverser.
- 02) Activer le **Moniteur** et proposer directement un **'&'** pour CRYPTER.
- 03) Frapper dans l'ordre **'a', 'b', 'c' ... 'x', 'y', 'z'** toutes les lettres dans l'ordre. En sortie nous devons retrouver la lettre proposée en entrée confirmant que deux permutations consécutives restituent la donnée de départ. Cette manipulation a pour but de s'assurer de la cohérence des deux tableaux, l'un pour le sens **Direct** et l'autre pour le sens **Réfléchi**.

➤ **Aller / Retour dans le Rotor de droite avec Indexation en Z-26.**

Dans cet exemple, le **Rélecteur** n'effectuant aucune permutation, on se retrouve dans le cas de la Fig.68 c'est à dire qu'en sortie on retrouvera la lettre donnée en entrée puisque le **Brouilleur** n'effectue aucune modification. C'est uniquement l'**Indexation** interne du **Rotor** qui a été changée. Pour changer cette fois on va **Indexer** sur la lettre **Z**. Cette manipulation a pour but de montrer la cohérence des traitements informatiques de l'**Indexage** dans le sens **Direct** et dans le sens **Réfléchi**. Ici aussi en sortie on doit retrouver la lettre frappée au clavier virtuel de la codeuse.

**MANIPULATIONS : (Suite)**

- 04) Modifier comme montré en Fig.69 le démonstrateur **P8** et le téléverser.
- 05) Activer à nouveau le **Moniteur** puis **"er"** pour modifier l'indexation.
- 06) Frapper d'erechef et dans l'ordre **'3', '2', '1', '1', '1', '26', 'a', 'a', 'a'** et **'b'** pour le **Rélecteur**.
- 07) Reprendre le CRYPTAGE avec **'&'**.
- 08) Recommencer dans l'ordre avec toutes les lettres de l'alphabet : **'a', 'b', 'c' ... 'x', 'y', 'z'** pour vérifier qu'elles ne sont pas modifiées en sortie du codage.

**OUPS !** J'ai totalement omis durant les exercices précédents de vous faire vérifier les transpositions en mode TEXTE. Je vous invite fortement dans cet esprit de reprendre les manipulations qui précèdent mais avant on va le faire ici pour vous montrer comment procéder :

- 09) Revenir au mode COMMANDE avec **'&'**.
- 10) Proposer **'t'** pour imposer le mode TEXTE.
- 11) Retourner en CRYPTAGE avec **'&'**.
- 12) Proposer la chaîne **"abcdefghijklmnopqrstuvwxy"**.

```
AAA IWQ GWY OAD
ABCDEF GHIJKL MNOPQR STUVWX YZ
>>> [A A A]
```

Fig.70

Quand on inscrit les 26 lettres en ordre alphabétique dans la fenêtre de saisie du **Moniteur** et que l'on valide, on obtient la copie d'écran de la Fig.70 comme résultat avec dans la zone rose la configuration de départ sous la fenêtre de la machine et en vert pastel la fin du **GROUPE d'identification**. Dans la zone bleue on retrouve inchangés les 26 caractères saisis qui sont regroupés par "paquets" de six lettres. On observe surtout que la visualisation sous la fenêtre semble inchangée. En réalité le **Rotor** de **Droite** a effectué 26 incréments, du coup on retrouve le **'A'**. En toute logique, chaque fois que le **Rotor** de **Droite** effectue un tour, il doit engendrer au moins une fois l'incrément du **Rotor** Central. Le mécanisme de rotation de la droite vers la gauche est assez complexe et n'est pas encore analysé ni traduit en langage C++. (*Chaque chose en son temps ...*)

**NOTE UTILE à ne surtout pas oublier :** Dans la suite de ce didacticiel nous allons inexorablement avoir à saisir de longues chaînes de caractères dans la fenêtre contextuelle du **Moniteur**. C'est volontairement que pour la manipulation en (12) j'ai inscrit l'intégralité de la suite des lettres de l'alphabet au lieu d'un raccourci du genre **"abc ... xyz"**. Ainsi, pour faciliter les exercices et vous éviter de commettre des erreurs, il vous suffira d'indexer le texte concerné, de le copier avec **[Ctrl] [c]**, de valider la fenêtre de saisie du **Moniteur** et enfin d'y recopier le texte source avec **[Ctrl] [v]**. À vous de penser ultérieurement à en profiter ...

Bien ; quand vous aurez repris les exercices qui précèdent en complétant les **Manipulations** avec les consignes du type de celles de (09) à (12), le moment d'envisager la **Rotation** des **Rotors** sera venu. C'est l'objet du chapitre qui "linéairement" fait suite à ce propos.

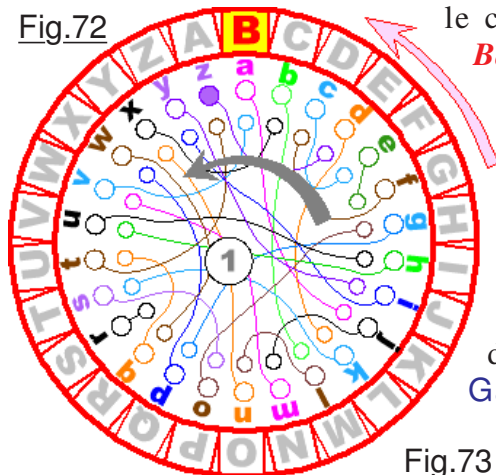
## 17) Émuler la rotation du Rotor de droite.

**M**atériellement un **Rotor** et sa **Bague extérieure** constitue une entité unique et indissociable. Il est évident que dans la pratique les deux tournent simultanément. Informatiquement on se doute que c'est totalement différent, car il y a des séquences de code pour gérer la rotation des liaisons électriques internes, et des routines indépendantes pour "faire tourner" la **Bague extérieure** sous la fenêtre de la machine. Nous avons vu dans les exercices précédents que la visualisation virtuelle du **Rotor** de **Droite** tourne à chaque caractère. Dans ce chapitre nous allons ajouter la rotation des circuits filaires internes du **Rotor** de **Droite**.

### ➤ **Rotation du Rotor de Droite indexé en A-1 dans le sens Direct.**

**T**ester cette facette du développement commence par modifier dans **P8** les paramètres du listage comme montré sur la Fig.71 pour ne valider que le sens **Direct**, et émuler la rotation interne du **Rotor** I supposé à **Droite** du **Brouilleur**. Dans

Fig.72



le cas analysé actuellement, la **Bague extérieure** tourne d'une

position, ainsi que le câblage interne du **Rotor**. On obtient la configuration de la Fig.72 avec les deux flèches courbes symbolisant les mouvements matériels. Il en résulte le tableau de la Fig.73 dans lequel les lettres minuscules présentent les couleurs des liaisons filaires virtuelles associées. (*Couleur des "gros cercles" relatifs aux contacts électriques d'Entrée.*) Dans la pratique, prendre en compte informatiquement la **Rotation** du câblage virtuel consiste à effectuer une **Rotation Logique à Gauche** d'un BIT pour les valeurs numériques du tableau qui intègre les écarts entre **Entrée** et **Sortie** sur le **Rotor** considéré.

Fig.73

Le résultat de cette **Rotation Logique** se retrouve dans la ligne

<b>Fenetre</b>	<b>B</b>	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
<b>Entrée</b>	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
<b>1</b> ←	9	10	2	7	1	23	9	13	16	3	8	2	9	10	18	7	3	0	22	6	13	5	20	4	10	4
<b>Sortie</b>	j	l	e	k	f	c	p	u	y	m	s	n	v	x	g	w	t	r	o	z	h	a	q	b	i	d

des valeurs tracées en bleu sur la Fig.73 représentant l'état du tableau qui en résulte.

### MANIPULATIONS :

- 1) Modifier comme montré en Fig.71 le démonstrateur **P8** et le téléverser.
- 2) Activer à nouveau le **Moniteur** puis "er" pour modifier l'indexation.
- 3) Maintenant c'est de la routine : Imposer dans l'ordre '3', '2', '1', '1', '1', '1', 'a', 'a', 'a' et 'b'.
- 4) Reprendre le manipulateur Morse virtuel avec '&'.
- 5) On teste à nouveau dans l'ordre toutes les lettres de l'alphabet : 'a', 'b', 'c' ... 'x', 'y', 'z' pour vérifier qu'en sortie du chiffage on obtient bien les permutations prévues dans le tableau.
- 6) Revenir au mode COMMANDE avec '&'.
- 7) Proposer 't' pour imposer le mode TEXTE.
- 8) Retourner en CRYPTAGE avec '&'.
- 9) Proposer la chaîne des 26 lettres "abcdefghijklmnopqrstuvwxyz".

**N**ous obtenons le résultat de la Fig.74 qui induit quelques explications. En effet, lorsque l'on frappe la première lettre 'a' elle est transposée en 'j' conformément aux prédictions du tableau dans sa ligne du bas. Puis, dès la frappe du caractère 'B' qui place la lettre 'C' sous la fenêtre de visualisation de la machine, au lieu d'obtenir un 'l', la permutation engendre un 'D'. Ce n'est pas du tout une erreur du programme. N'oublions pas que chaque frappe d'une lettre engendre une rotation virtuelle de plus pour le **Rotor** et la ligne des décalages en bleu se "déporte" à chaque fois d'une position vers la gauche dans le tableau représentatif du composant. Il est évident qu'il



```

Une lettre -> [A A B] >>> [A] devient [J] .---
Une lettre -> [A A C] >>> [B] devient [D] -..
Une lettre -> [A A D] >>> [C] devient [D] -..
Une lettre -> [A A E] >>> [D] devient [M] --
Une lettre -> [A A F] >>> [E] devient [U] ..-
Une lettre -> [A A G] >>> [F] devient [N] -.
Une lettre -> [A A H] >>> [G] devient [P] .---.
Une lettre -> [A A I] >>> [H] devient [Z] ---.
Une lettre -> [A A J] >>> [I] devient [L] -..
Une lettre -> [A A K] >>> [J] devient [F] ..-
Une lettre -> [A A L] >>> [K] devient [X] ---.
Une lettre -> [A A M] >>> [L] devient [F] ...
Une lettre -> [A A N] >>> [M] devient [W] --
Une lettre -> [A A O] >>> [N] devient [W] --
Une lettre -> [A A P] >>> [O] devient [Q] ---.
Une lettre -> [A A Q] >>> [P] devient [Q] ---.
Une lettre -> [A A R] >>> [Q] devient [Z] ---.
Une lettre -> [A A S] >>> [R] devient [H] ....
Une lettre -> [A A T] >>> [S] devient [A] -.
Une lettre -> [A A U] >>> [T] devient [C] ---.
Une lettre -> [A A V] >>> [U] devient [M] --
Une lettre -> [A A W] >>> [V] devient [Y] ---.
Une lettre -> [A A X] >>> [W] devient [S] ...
Une lettre -> [A A Y] >>> [X] devient [K] -.
Une lettre -> [A A Z] >>> [Y] devient [S] ...
Une lettre -> [A A A] >>> [Z] devient [J] .---
Retour au mode COMMANDES.

```

```

COMMANDE -> [T]
Passage au mode TEXTE. (60 caracteres MAXI.)
COMMANDE -> [&]
Passage au mode CRYPTAGE.
AAA IWQ GWY OAD
JDDMUN PZLFXF WWQOZH ACMYSK SJ
>>> [A A A]

```

Fig.74

n'est pas suffisant de se contenter de la première lettre pour vérifier le démonstrateur. Aussi, disposant d'un simulateur tel que celui de la Fig.75 mais animé, j'ai été en mesure de tester les 26 lettres pour m'assurer du bienfondé de l'algorithme qui gère la **Rotation** dans le sens **Direct**. (Vous vous doutez que cet artifice a également été employé pour le sens **Réfléchi**.) Par exemple pour la lettre **P** sous la fenêtre de visualisation on peut vérifier sur la Fig.75 que le 'o' en **Entrée** devient bien un 'q' en **Sortie**.

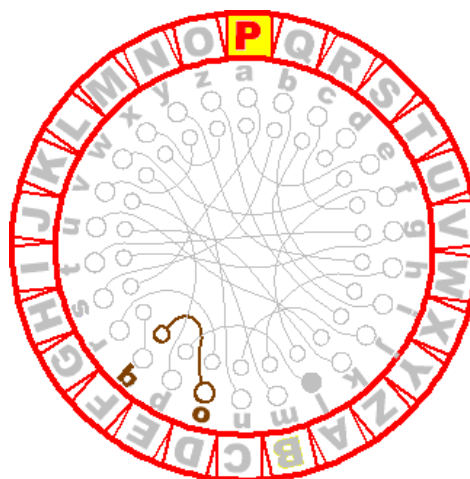


Fig.75

Pour résumer, sur la Fig.74 la colonne verte représente le changement à chaque caractère de la lettre située sous la fenêtre de visualisation d'Enigma. Dans la colonne rose sont alignées verticalement les lettres frappées en **Entrée** et dans la zone bleue leur permutation par le cryptage. Dans les consignes "en jaune" on passe au cryptage en mode TEXTE. On retrouve en violet le **GROUPE d'identification** et en "bleu clair" les transpositions formatées en "paquets" de six caractères. Enfin en "orange" la fenêtre de visualisation se retrouve en **[AAA]** car le **Rotor** de Droite à tourné de 26 incréments.

### ➤ Rotation du Rotor de Droite indexé en A-1 dans le sens Réfléchi.

C'est un peu comme du "rabâchage", ou reprendre presque les mêmes et refaire encore et encore. Vous devez commencer à deviner la suite qui consiste à modifier une nouvelle fois les paramètres dans **P8**. Autant dire que grâce à ces booléens faciles à changer, nous l'aurons bien rentabilisé ce démonstrateur.

### MANIPULATIONS :

- 01) Modifier comme montré en Fig.76 le démonstrateur **P8** et le téléverser.
- 02) Reprendre directement la transmission Morse virtuelle avec '&'.
- 03) On teste encore dans l'ordre toutes les lettres de l'alphabet : 'a', 'b', 'c' ... 'x', 'y', 'z'.
- 04) Revenir au mode COMMANDE avec '&'.
- 05) Proposer 't' pour imposer le mode TEXTE.
- 06) Retourner en CRYPTAGE avec '&'.
- 07) Proposer une nouvelle fois la chaîne "abcdefghijklmnopqrstuvwxyz". On doit obtenir le résultat de la Fig.77 en remarquant que seule la première transposition du 'a' en 'V' correspond à celle du tableau.

```

#define Direct false
#define Reflechi true
#define Indexer_en_Direct false
#define Indexer_en_Reflechi false
#define Rotation_en_Direct false
#define Rotation_en_Reflechi true

```

Fig.76

```

COMMANDE -> [T]
Passage au mode TEXTE. (60 caracteres MAXI.)
Attention : LGR > MAX.
COMMANDE -> [&]
Passage au mode CRYPTAGE.
AAA IWQ GWY OAD
VEALUY DLOBXE WIRNYH LQYBOK RJ

```

Fig.77

## 18) Installer et vérifier les Rotors II, III, IV et V.

Intrinsèquement chaque élément qui constitue le brouilleur a été écrit et testé dans les démonstrateurs précédents. Maintenant il va falloir commencer à mettre en cascade trois **Rotors** parmi les cinq sur la machine virtuelle. Pour le moment *on ne va que vérifier les quatre rotors qui sont logés en EEPROM avec la nouvelle approche*. Toutefois lors de l'initialisation, le nouveau démonstrateur **P09\_Mise en place\_des\_Rotors.ino** se charge également de *placer dans l'ordre en virtuel les trois rotors précisés en initialisation* de la machine. Pour conserver les facilités apportées par les paramétrages utilisés lors des manipulations précédentes, **P9** est directement construit à partir de **P8** et *pour le cryptage ne prend en charge que le Rotor de Droite*. En principe on suppose ici que vous avez téléversé et activé **P00C** dans le chapitre précédent.

➤ **Installer le Rotor II indexé en A-1 à Droite et tester en sens Direct.**

Exactement comme pour **P8** les paramètres en tête de listage sont prévus pour commencer cette vérification directement sans avoir à le modifier. Il est temps d'aller ouvrir dans le dossier **<Documents>** joint à ce didacticiel le fichier **FICHES.pdf** et éventuellement imprimer les fiches au format **A5** qu'il contient. Ces dernières étant éventuellement plastifiées et découpées, c'est la **Fiche n°2** et la **Fiche n°3** qu'il faut avoir sous la main pour valider le **Rotor II**.

### MANIPULATIONS :

- 01) Téléverser **P09\_Mise en place\_des\_Rotors.ino** et activer le **Moniteur de l'IDE**.
- 02) Frapper un **"er"** pour initialiser les **Rotors**.
- 03) Indiquer **'4', '3', '2', '1', '1', '1', 'a', 'a', 'a'**, et enfin **'b'** pour le **Réflexteur**.
- 04) Proposer ensuite **'s'** suivi de **'o'** pour enregistrer cette configuration en EEPROM.
- 05) Passer en mode CRYPTAGE avec **'&'**.
- 06) Tester dans l'ordre toutes les lettres de l'alphabet : **'a', 'b', 'c' ... 'x', 'y', 'z'** pour vérifier qu'en sortie du chiffage on obtient bien les permutations prévues dans le tableau du **haut** de la **Fiche n°2**.
- 07) Revenir au mode COMMANDE avec **'&'**.
- 08) Proposer **'t'** pour imposer le mode TEXTE.
- 09) Retourner en CRYPTAGE avec **'&'**.
- 10) Proposer la chaîne des 26 lettres **"abcdefghijklmnopqrstuvwxy"**.

➤ **Installer le Rotor II indexé en A-1 à Droite et tester en sens Réfléchi.**

Copiant impunément sur les exercices du chapitre précédent, pour tester le sens Réfléchi il suffit de refaire les même manipulation, mais d'avoir au préalable modifié comme montré sur la Fig.78 les déclarations en tête de **P09**.

```
#define Direct false
#define Reflechi true
#define Indexer_en_Direct false
#define Indexer_en_Reflechi false
#define Rotation_en_Direct false
#define Rotation en Reflechi false
```

Fig.78

### MANIPULATIONS : (Suite)

- 11) Activer le **Moniteur de l'IDE**.
- 12) Tester dans l'ordre toutes les lettres de l'alphabet : **'a', 'b', 'c' ... 'x', 'y', 'z'** pour vérifier qu'en sortie du chiffage on obtient bien les permutations prévues dans le tableau du **bas** de la **Fiche n°2**.
- 13) Reprendre le mode COMMANDE avec **'&'**.
- 14) Imposer **'t'** pour valider l'option TEXTE.
- 15) Reprendre le CRYPTAGE avec **'&'**.
- 16) "Copier / Coller" la chaîne des 26 lettres **"abcdefghijklmnopqrstuvwxy"**.

**Remarque sur ce démonstrateur :** Contrairement à toutes les autres procédures du programme, la routine **Installe\_les\_rotors\_et\_le\_Reflecteur** est beaucoup trop longue et contrairement aux conseils donnés dans ce didacticiel elle dépasse largement la hauteur de visualisation de l'écran. C'est au détriment de la programmation si pour une quelconque raison il faudrait la modifier. Pour éviter cet inconvénient il conviendrait de la scinder en plusieurs procédures effectuant chacune un travail spécifique. Par exemple regrouper la mise en place des Rotors, regrouper leur Indexation etc. Ce n'est pas ce qui est fait ici car l'ensemble est relativement "linéaire" et reste d'une lisibilité acceptable. Faire plusieurs procédures augmenterait la taille du programme et certaines variables locales deviendraient globales. C'est un compromis ...

➤ **Installer le Rotor III indexé en A-1 à Droite et tester en sens Direct.**

Sans que ce ne soit obligatoire, je vous propose en fin de ce chapitre de tester entièrement les **Rotors** logés en EEPROM, et ce dans les deux sens d'utilisation. Si ces exercices assez répétitif il faut bien l'avouer "vous barbent", n'hésitez-pas à passer directement au chapitre 18. Comme on va commencer par tester les trois autres **Rotors** dans le sens **Direct**, il faut revenir à l'état initial de **P9** dont la Fig.79 en rappelle la combinatoire logique des paramètres. La **Fiche n°3** en main on peut passer à la vérification du **Rotor III** que l'on doit placer à **Droite** sur la machine.

```
#define Direct true
#define Reflechi false
#define Indexer_en_Direct false
#define Indexer_en_Reflechi false
#define Rotation_en_Direct false
#define Rotation_en_Reflechi false
```

Fig.79

**MANIPULATIONS :** (Suite)

- 17) Téléverser **P9** modifié conformément à la Fig.79 et activer le **Moniteur** de l'**IDE**.
- 18) Frapper un "er" pour initialiser les **Rotors**.
- 19) Indiquer '1', '2', '3', '1', '1', '1', 'a', 'a', 'a', et enfin 'c' pour le **Réflexteur**.
- 20) Proposer ensuite 's' suivi de 'o' pour enregistrer cette configuration en EEPROM.
- 21) Passer en mode CRYPTAGE avec '&'.
- 22) Tester dans l'ordre toutes les lettres de l'alphabet : 'a', 'b', 'c' ... 'x', 'y', 'z' pour vérifier qu'en sortie du chiffage on obtient bien les combinaisons prévues dans le tableau du *haut* de la **Fiche n°3**.

➤ **Installer le Rotor IV indexé en A-1 à Droite et tester en sens Direct.**

- 23) Revenir en mode COMMANDES avec '&'.
- 24) Consigner un "er" pour initialiser les **Rotors**.
- 25) Proposer '2', '3', '4', '1', '1', '1', 'a', 'a', 'a', et enfin 'c' pour le **Réflexteur**.
- 26) Reprendre le mode CRYPTAGE avec '&'.
- 27) Manipuler dans l'ordre toutes les lettres de l'alphabet : 'a', 'b', 'c' ... 'x', 'y', 'z' pour vérifier qu'en sortie du codage on obtient bien les résultats prévues dans le tableau du *haut* de la **Fiche n°4**.

➤ **Installer le Rotor V indexé en A-1 à Droite et tester en sens Direct.**

**T**ravail routinier ... vous étiez prévenus. Du reste dans les manipulations on a bien un sentiment de Copier/Coller". *C'est incontournable, si l'on veut un jeu d'essais complet.* Vous pouvez en faire l'impasse, en espérant que mes vérifications seront totales et vraiment valides. Pour ma part je suis obligé de tout valider dix fois plutôt qu'une, car, lorsque l'on va chaîner tous les éléments, si en sortie le résultat n'est pas correct, la combinatoire des erreurs possibles sera explosive. La rigueur et la méthode sont plus que jamais indispensables.

- 28) Repasser en mode COMMANDES avec '&'.
- 29) Répéter "er" pour initialiser les **Rotors**.
- 30) Proposer '3', '4', '5', '1', '1', '1', 'a', 'a', 'a' suivi de 'b' pour le **Réflexteur**.
- 31) Adopter à nouveau le CRYPTAGE avec '&'.
- 32) Transmettre dans l'ordre toutes les lettres de l'alphabet : 'a', 'b', 'c' ... 'x', 'y', 'z' pour vérifier qu'en éclairage des ampoules virtuelles on a bien les lettres prévues dans le tableau du *haut* de la **Fiche n°5**.

➤ **Installer le Rotor III indexé en A-1 à Droite et tester en sens Réflexi.**

- 33) Téléverser **P9** modifié conformément à la Fig.78 et activer le **Moniteur** de l'**IDE**.
- 34) Imposer "er" pour initialiser une fois de plus les **Rotors**.
- 35) Indiquer '1', '2', '3', '1', '1', '1', 'a', 'a', 'a' puis 'b' pour le **Réflexteur**.
- 36) Passer en CRYPTAGE avec '&'.
- 37) Transmettre dans l'ordre toutes les lettres de l'alphabet : 'a', 'b', 'c' ... 'x', 'y', 'z' pour vérifier qu'en permutations on obtient bien les données prévues dans le tableau du *bas* de la **Fiche n°3**.



Ben Môa môa, je trouve que tous ces Rotors c'est une affaire qui tourne !



➤ Installer le **Rotor IV** indexé en A-1 à Droite et tester en sens **Réfléchi**.

**MANIPULATIONS :** (Suite)

- 38) Revenir en mode COMMANDES avec '&'.
- 39) Frapper "er" pour initialiser les **Rotors**.
- 40) Indiquer '2', '3', '4', '1', '1', '1', 'a', 'a', 'a', et enfin 'b' pour le **Rélecteur**.
- 41) Reprendre le mode CRYPTAGE avec '&'.
- 42) Taper dans l'ordre toutes les lettres de l'alphabet : 'a', 'b', 'c' ... 'x', 'y', 'z' pour vérifier qu'en sortie du chiffage on obtient bien le cryptage prévu dans le tableau du **bas** de la **Fiche n°4**.
- 43) Retrouver le mode COMMANDES avec '&'.

➤ Installer le **Rotor V** indexé en A-1 à Droite et tester en sens **Réfléchi**.

- 44) Imposer "er" pour initialiser une "dernière" fois les **Rotors**.
- 45) Indiquer '3', '4', '5', '1', '1', '1', 'a', 'a', 'a' et 'b' pour finir.
- 46) Passer en CRYPTAGE avec '&'.
- 47) Proposer dans l'ordre toutes les lettres de l'alphabet : 'a', 'b', 'c' ... 'x', 'y', 'z' pour vérifier qu'en sortie de la machine on obtient bien le codage prévu dans le tableau du **bas** de la **Fiche n°5**.
- 48) Par discipline repasser en COMMANDES avec '&'.
- 49) Pour être complet donner 't' pour valider l'option TEXTE suivi de '&'.
- 48) Enfin "Copier / Coller" la chaîne des 26 lettres "**abcdefghijklmnopqrstuvwxyz**".
- 48) Sortir du chiffage avec '&'.

Ce dernier test de validation s'achève sur le résultat dont la Fig.80 montre une copie d'écran partielle. C'est fait, notre magasin contient tous les éléments d'Énigma dont nous aurons besoin, et ils sont "certifiés conformes". En principe, sauf catastrophe imprévue, en principe le contenu de l'EEPROM est désormais figé. Il est détaillé dans la **Fiche n°10** pour le bas de la mémoire, c'est à dire les textes du dialogue H/M et dans la **Fiche n°11** pour le haut contenant les **Rotors** et les deux **Rélecteurs**.

```

Une lettre -> [A A B] >>> [A] devient [Q]  --.-
Une lettre -> [A A C] >>> [B] devient [C]  --.-
Une lettre -> [A A D] >>> [C] devient [Y]  --.-

Une lettre -> [A A Y] >>> [X] devient [P]  ---.
Une lettre -> [A A Z] >>> [Y] devient [H]  ....
Une lettre -> [A A A] >>> [Z] devient [B]  -...
Retour au mode COMMANDES.
COMMANDE -> [T]
Passage au mode TEXTE. (60 caracteres MAXI.)
COMMANDE -> [&]
Passage au mode CRYPTAGE.
AAA IWQ GWY OAD
QCYLXW ENFTZO SMVJUD KGIARP HB
>>> [A A A]
Retour au mode COMMANDES.
COMMANDE ->

```

Fig.80

➤ Un bilan sur l'évolution du programme.

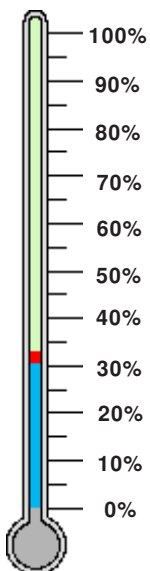


Fig.81

Avec l'émulation des cinq **Rotors**, leur **Indexation** et la procédure de **Rotation** de celui qui sera placé à **Droite**, on a franchi une étape qui est loin d'être dérisoire. La mise en place des trois **Rotors** avec leur **Indexation** ainsi que la **Rotation** systématique de l'élément de droite fait passer le programme de 9740 octets soit 31 % de l'espace réservé au programme occupé à 10206 octets soit 33% de consommation de place. L'augmentation de température sur le thermomètre fictif reste assez dérisoire et l'on ne peut qu'envisager l'avenir avec sérénité. Il est certainement probable qu'au final il restera une bonne partie de la zone réservée au code inutilisée. Il serait-donc possible de se laisser glisser vers la facilité. C'est totalement contraire à l'approche que l'on s'est fixé dans ces lignes. Aussi, bien que ce ne soit certainement pas vraiment nécessaire, nous allons continuer à optimiser à outrance le code ne serait-ce que pour "faire de la belle ouvrage". Pour clore ce chapitre, *le filtrage en initialisation n'a pas tenu compte du fait que l'opérateur peut actuellement mettre en place deux fois ou trois fois le même Rotor. Sur la codeuse réelle ce n'était naturellement pas possible. Aussi, dans notre contexte virtuel il va falloir absolument parer ce type d'erreur de logique* ce qui va induire encore du dialogue H/M et plus de code objet. C'est là que l'on sera content d'avoir optimisé notre C++ à tous les niveaux.

## 19) Concrétiser le Brouilleur complet.

Avant d'envisager le brouilleur totalement "assemblé" avec à gauche la présence du **Réfecteur** nous allons intercaler une étape "prudente" qui va consister à ne placer que trois des cinq **Rotors** dans le **Brouilleur** virtuel et de vérifier leur traversée dans les deux sens. C'est **P10\_Le\_Brouilleur\_complet.ino** qui va servir à cette expérimentation intermédiaire. Toutefois, comme envisagé en bas de la page précédente, nous allons au préalable compléter le filtrage des **Rotors** à leur installation virtuelle pour s'assurer que le même élément ne soit pas choisi deux fois.

➤ **Impossible de placer deux fois le même Rotor dans le Brouilleur.**

C'était promis en fin du chapitre précédent, nous allons effectuer le filtrage lors de l'initialisation de la Machine pour ne pas installer deux fois le même **Rotor** dans le **Brouilleur**, erreur impossible à commettre avec la machine réelle, mais tout à fait possible avec les éléments virtuels qui peuvent se dupliquer à l'infini.

### MANIPULATIONS :

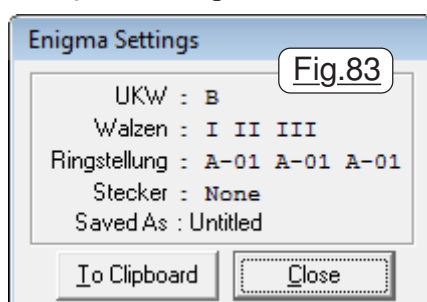
- 01) Téléverser **P10\_Le\_Brouilleur\_complet.ino** et activer le **Moniteur** de l'**IDE**.
- 02) Frapper un "er" pour initialiser le **Brouilleur**.
- 03) Indiquer '4' pour le **Rotor** de gauche. (*À ce stade les cinq sont possibles.*)
- 04) Proposer '4' pour le **Rotor** du centre. Refus du programme car il n'est plus disponible.
- 05) Indiquer '5' par exemple qui se trouve encore dans le coffret virtuel.
- 06) Puis, pour le **Rotor** de droite choisir le '4'. (*Encore une erreur !*)
- 07) Suite à une étourderie peu crédible, cette fois c'est '5' qui sera refusé. (*En résumé il sera impossible d'utiliser deux fois le même élément, on ne sortira de cette fonction que lorsque l'on aura proposé au programme exclusivement que des options cohérentes.*)
- 08) Pour le plaisir de consolider notre savoir-faire, terminer cette initialisation à votre guise.

➤ **Confronter le démonstrateur à la réalité.**

Jusqu'à présent, à partir de beaux tableaux tels que ceux de la Fig.33, de la Fig.36 ou encore celui de la Fig.42 nous avons élaborée des éléments virtuels que nous avons testé avec rigueur dans des démonstrateurs. Vous vous doutez bien que ces tableaux sont issus d'informations puisées sur la toile. Avant de m'en servir pour coder en C++, j'ai vérifié la validité de ces informations en comparant sur plusieurs sources. Puis, avec **P10\_Le\_Brouilleur\_complet.ino** on a assemblé virtuellement tous ces éléments pour aboutir à un **Brouilleur** presque complet. (*Presque, car la rotation des trois rotors n'est pas encore implémentée.*) Pour tester le comportement de ce prototype virtuel, il nous faut impérativement disposer d'une vraie machine Énigma. Une telle codeuse n'est pas forcément dans nos moyens, car les prix de ces antiquités sont absolument astronomiques. Aussi nous allons utiliser un simulateur reproduisant avec rigueur le comportement de l'unité réelle. Il en existe de nombreuses versions sur Internet toutes présentant des fonctionnements fiables. Comme il faut bien en choisir une nous allons utiliser celle de :

<https://www.ciphermachinesandcryptology.com/en/enigmasim.htm>

Facile à installer, lorsque l'on active ce simulateur, nous obtenons l'affichage qui ressemble à celui de la Fig.84 avec au départ l'initialisation par défaut de la Fig.83 que nous allons utiliser pour tester le démonstrateur **P10**. Pour quitter cette application il suffit de cliquer en **16** avec le bouton gauche de la souris. (*Le programme nous demande si l'on désire sauvegarder la configuration si cette dernière a été modifiée.*) Cliquer en **1** ouvre le menu de la Fig.82 avec l'affichage de la Fig.83 si on



valide l'option jaune, avec la possibilité de sauvegarder ou de recharger une configuration avec les commandes bleues, voir de sortir par la consigne verte etc. Quand on clique sur une touche du clavier **12** comme en **11** par exemple, elle devient plus petite simulant son enfoncement. Le **Rotor** de droite en **3** tourne en faisant un petit bruit caractéristique, puis l'une des ampoules du tableau **9** en **10** par exemple s'illumine en fonction du cryptage.

Fig.82

Clipboard
Auto Typing
View Key
Load Key
Save Key
Clear Key
Gallery
About
Help
Exit



Lorsque l'on relâche le bouton de la souris, alors elle s'éteint. Quand le **Rotor** de droite a effectué une rotation, il entraîne en **8** son voisin du centre. Puis, ce sera le tour en **7** du **Rotor** le plus à gauche. Machine "fermée" dans l'état de celle de l'exemple on peut manuellement positionner à convenance chaque **Rotor** en cliquant soit en **2** sur le haut de la couronne dentée, soit en **5** vers le bas des pétales de la "marguerite". Cliquer sur le verrouillage **6** nous engage dans une promenade technique de découverte des éléments de la machine. C'est en cliquant sur le verrouillage **4** que l'on ouvre le plateau du dessus pour installer et configurer le **Brouilleur**. (*Nous y reviendrons dès le chapitre 20.*) Commodité apportée par la version virtuelle, le fait de cliquer sur la serrure **13** ouvre la zone **14** dans laquelle vont s'écrire le texte d'origine et celui crypté avec formatage par groupe de cinq lettres. Par exemple en **15** on a sur la ligne du haut la première lettre 'A' frappée au clavier en **11** et en dessous son codage en 'B'. Cet artifice remplace en pratique celui de la feuille de papier sur laquelle

l'opérateur inscrirait son message avant de le transmettre en morse, ou au contraire de le décoder. Il est évident que ce simulateur comme le notre est prévu pour pouvoir changer les **Rotors** ainsi que leur initialisation et remplacer le **Réfecteur** utilisé. C'est en cliquant dans la zone des prises du tableau des fiches vu ici par le dessus en **17** que l'on peut combiner les dix inversions filaires. Ces manipulations seront détaillées en temps utile. Il reste bien d'autres détails d'utilisation de ce logiciel astucieux, vous les découvrirez par vous-même lors de nos expérimentations.

### ➤ Les informations sur Internet.

**S'** il est un thème qui a fait couler beaucoup d'encre sur son sujet, c'est bien Énigma. Que ce soit dans les archives papier ou en "dématérialisé" sur la toile, les articles décrivant cette machine ainsi que son historique sont légion. Il faut toutefois "pondérer" ces explications et parfois les contextualiser. Par exemple, quand cette codeuse est explicitée, on peut lire, *et dans plusieurs sources*, que : "**Comme la roue de droite tourne à chaque frappe sur le clavier, la même lettre sera différente durant 26 saisies.**". Cette affirmation péremptoire n'est pas totalement exacte. S'il est vrai qu'au cours de ces rotations la combinatoire de substitution change, il peut arriver que la lettre soit codée de façon identique plusieurs fois durant un cycle. Considérons la Fig.85 issue du simulateur que nous avons adopté pour sa fiabilité et sa simplicité de mise en œuvre. (*Ce dessin est compacté pour ne montrer que les zones pertinentes.*) Initialement les trois **Rotors** sont sur la position **A**. Dans cet exemple on a frappé vingt fois la lettre 'A', c'est à dire que le **Rotor** n'a pas encore fait un tour complet et n'a pas entraîné le **Rotor** central. Pourtant on constate qu'à de nombreuses reprises on retrouve un cryptage déjà effectué. Les informations proposées dans les exposés ne sont pas franchement inexacts, mais elles sont forcément "simplifiées" pour des raisons pédagogiques. Donc, parfois il ne faudra pas s'étonner d'une divergence entre ces explications et les résultats obtenus sur des simulateurs fiables reproduisant exactement le comportement de la machine d'origine.





## ➤ Vérifier le Brouilleur.

Opération capitale avant d'envisager la mise en œuvre des mécanismes qui engendrent les incréments des deux autres **Rotors**. Dans ce but **P10** a été téléversé dans l'exercice précédent, la fenêtre du **Moniteur de l'IDE** est ouverte. Le simulateur d'Énigma de la Fig.84 est également activé. Les ingrédients sont réunis, on peut effectuer la vérification complète de notre version actuelle de la chiffreuse. (Voir en Fig.90 la remarque de la petite **salamandre** à ce sujet ...)

### MANIPULATIONS :

- 01) Pour la forme "repartir à zéro" avec un RESET.
- 02) Vérifier sur [Enigma] que nous avons bien la configuration de la Fig.83 ou dans le menu de la Fig.82 imposer **Clear Key** suivi de **OK**. (Dans ces exercices tout ce qui concerne le simulateur glané sur Internet servant de référence sera en marron.)
- 03) Frapper en **11** sur la touche 'A' du clavier virtuel. Dans l'état actuel d'[Enigma] son ampoule en 'B' s'illumine. Au relâcher du clic de souris elle s'éteint et la fenêtre affiche [AAB].
- 04) Cliquer en **13** pour "étaier" notre feuille de papier virtuelle. Pas de surprise, on retrouve en **15** sur la ligne du haut la lettre proposée à la machine et en ligne du bas sa transposition.
- 05) Cliquer sur le clavier en **12** les touches immatérielles 'B', 'C', 'D', 'E' et 'F'. À ce stade la machine de vérification affiche [AAG] et [BJELR Q].
- 06) Passons en fenêtre contextuelle du **Moniteur** et frapper un '&' pour chiffrer ce texte initial.
- 07) Proposer 'a', 'b', 'c', 'd', 'e' et 'f'. (Penser à valider à chaque lettre.) On obtient à chaque fois un descriptif tel que celui de la Fig.86 qui précise le "cheminement" du courant électrique dans le **Brouilleur** avec en vert la lettre proposée et en rose son cryptage en sortie sur le **Collecteur**. Dans la zone orange l'état actuel de la fenêtre et en jaune le **Rélecteur**.

Fig.86

Cheminement :

```

F [f < D] [d < C] [c < A] <- A
S [s > S] [s > E] [e > B] -> B
[A A B] >>> [A] devient [B] -...
Une lettre ->
  
```

La flèche bleue indique le cheminement à travers III, puis II et enfin I avec en MAJuscule la lettre en entrée et en MINuscule celle en sortie. La flèche violette traduit le retour depuis le **Rélecteur** en retraversant dans l'ordre I, puis II et enfin III. Dans ce tout premier cheminement la lettre est devenue dans l'ordre A > c > d > f > F > S > S > E > B > B. Il me semble utile pour ce premier test de suivre ce cheminement et le comparer sur les tableaux du didacticiel représentant les éléments du **Brouilleur**.

**ATTENTION :** Ne pas oublier que le **Rotor** de droite **s'incrémente** quand on enfonce la touche **avant le cryptage** de cette dernière.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
C	E	G	I	K	B	O	Q	S	W	U	Y	M	X	D	H	V	F	Z	J	L	R	P	N	A	

Fig.88

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Z	F	A	O	B	R	C	P	D	T	E	U	M	Y	G	X	H	W	V	K	Q	J	N	L	S	

Fig.89

Pour tenir compte de la remarque de l'encadré, on ne peut utiliser directement la table de la Fig.36 pour le **Rotor III**. Dans ce but sur la Fig.87 ce dernier à tourné d'une position ce qui permet de construire la table pour le sens direct en Fig.88 et celle en Fig.89 pour le sens réfléchi. Pour le moment dans toutes les tables les trois **Rotors** son **Indexés** sur la bague **A-1**. On peut passer à la vérification sur papier du cheminement affiché par **P10** pour la première touche frappée virtuellement.

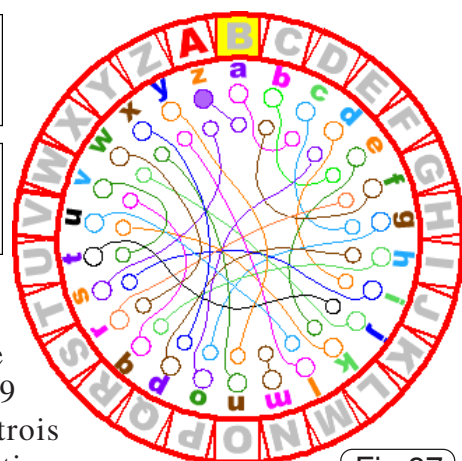


Fig.87



Fig.90

Hé les amis(es), il faut utiliser la configuration de la Fig.83 pour décrypter et ne pas donner le point d'exclamation.

ALOOA WOQLW NEICT OTTYI LVALG  
 KOACS ACOYD EXXKD GNSNO GMIOR GFZZN  
 UDNUD VCTSQ XZETH BCBLLH SFMYO HILNF  
 YKPNE NMSD !



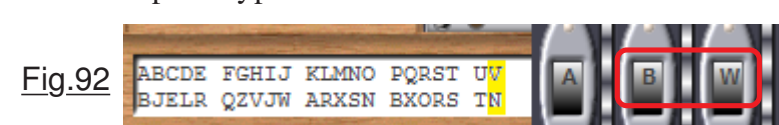
Partant du haut vers le bas on transite en 1 dans le sens direct et en 2 dans le sens réfléchi. On traverse le **Rotor III** en **A**, puis le **Rotor II** en **B** et enfin le **Rotor I** en **C**. Comme indiqué par le démonstrateur, la lettre **A** devient **C**, puis **D** et enfin **F**. Dans le **Réfecteur** ce **F** est permuté en **S** conformément au tableau du centre repéré en jaune. Puis le cheminement se poursuit en sens inverse de la gauche vers la droite. En sens réfléchi le signal traverse le **Rotor I** en **E**, puis le **Rotor II** en **F** et enfin le **Rotor III** en **G**. La lettre **S** devient **S**, (*Inchangée en E.*) puis **E** et **B** en sortie du **Collecteur**. Le cheminement affiché par le programme est donc conforme à l'étude théorique sur papier, les algorithmes utilisés semblent conformes. Pour les valider il convient de compléter notre jeu d'essais.

#### MANIPULATIONS : (Suite)

08) Sur [Enigma] proposer les autres lettres de l'alphabet jusqu'à 'Z'.

09) Puis dans le champ de saisie du Moniteur faire de même : Lettres de 'g' à 'z'.

Avec les 21 premières lettres de l'alphabet il y a correspondance parfaite entre le comportement de l'[Enigma] servant de référence. Mais à partir de la lettre 'v' il y a divergence. Rassurez-vous, ce ne sont pas les algorithmes de P10 qui sont en cause, mais son développement incomplet. En effet, sur la machine d'origine, le **Rotor III** entraine son **Rotor** situé à sa gauche lorsqu'il passe de la lettre **V** à la lettre **W** sur sa bague extérieure, ce que montre bien la Fig.92 dans l'encadré rouge, alors que sur notre prototype immatériel cette incrémentation n'est pas encore émulée. Rien d'alarmant en



définitive. Toutefois, avant de compléter le logiciel on va encore effectuer des tests pour valider notre **Brouilleur**.

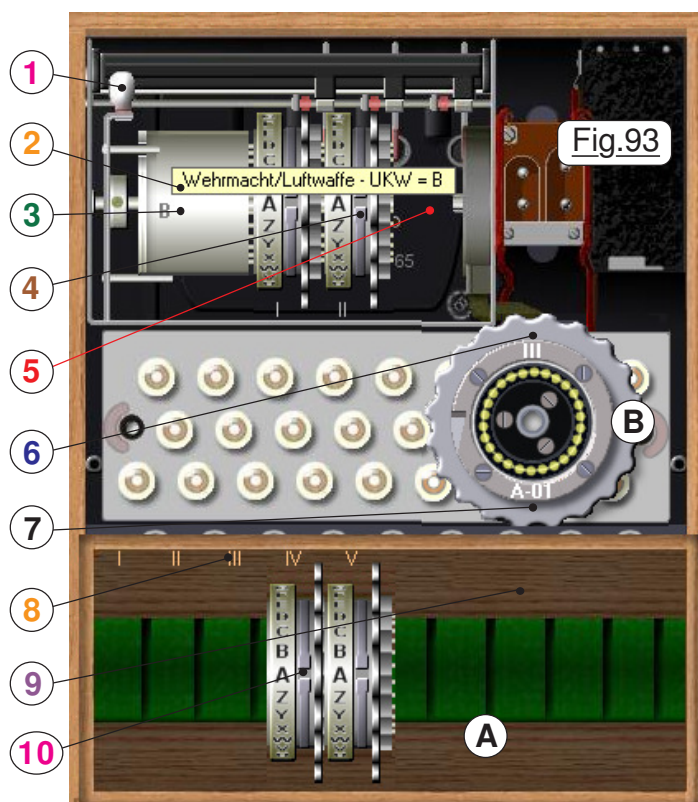


## 20) Un jeu d'essais complet pour valider le Brouilleur.

Continuer à développer le projet va consister à émuler la rotation sur les trois rotors, puis faire intervenir le tableau des fiches croisées. Chaque étape subira des tests sévères pour en valider les algorithmes. Hors, il est probable qu'à certains moments on va constater des problèmes. Pour les résoudre il faut absolument s'assurer que les divergences sont bien le fait des nouveaux éléments ajoutés à la machine. En d'autres termes, à ce stade de notre entreprise ludique, il faut absolument nous convaincre que le **Brouilleur** dans son ensemble est irréprochable.

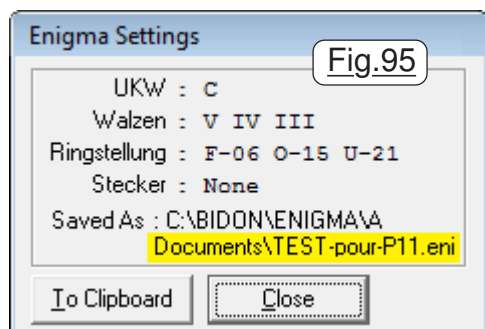
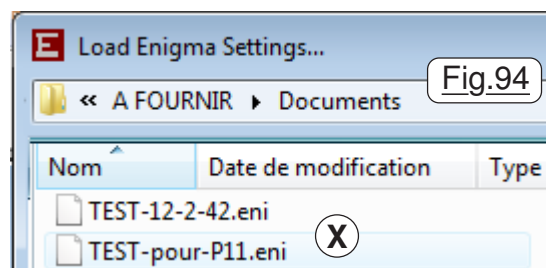
### ➤ **Modification de la configuration de l'[Enigma] de référence.**

Comparer le comportement des deux entités virtuelles suppose on s'en doute un peu d'avoir une initialisation identique sur les deux codeuses. Cet impératif nous oblige à voir comment procéder sur **[Enigma]**. Un moyen radical d'effectuer un test significatif consiste à chambouler intégralement la combinatoire d'une initialisation en adoptant un joyeux brassage des divers paramètres possibles. Commencer par cliquer sur le verrouillage **4** de la Fig.84 pour ouvrir le plateau



du dessus et accéder comme sur la Fig.93 aux éléments internes de la chiffreuse. Cliquer n'importe où sur l'un des **Rotors** en **4** le dépose du **Brouilleur** et le place en **B**. C'est dans cette configuration que si l'on clique en **6** ou en **7** on en modifie à convenance l'**Indexation** interne. En cliquant dans le coffret de rangement en **9** le **Rotors** déposé en **B** est automatiquement remplacé dans son logement repéré en **8**. Pour installer un **Rotor** sur le **Brouilleur** on procède à l'inverse. On clique sur ce dernier en **10** pour le placer en **B**. Dans cette position on l'**Indexe** en interne avec **6** et **7**. Puis on clique dans un emplacement libre en **5** pour l'installer sur le **Brouilleur**. C'est dans la zone restreinte **3** que l'on choisit le type de codeuse et le **Réflecteur** installé. Bien faire attention à sélectionner **Wehrmacht/luftwaffe - UKW = B** ou **C** indiqué en **2** en fonction de votre choix. On peut alors refermer le plateau en cliquant en **1** et orienter les **Rotors** comme précisé en Fig.84 de la page 47. Noter au passage que l'on ne peut pas

refermer le plateau s'il n'y a pas trois **Rotors** installés sur le **Brouilleur**. Pour effectuer notre jeu d'essais avec **P11** il faut impérativement installer la configuration de la Fig.94 soit par manipulations directe de la machine virtuelle, soit en rechargeant l'initialisation disponible dans le dossier **<Documents>** par la commande **Load key** du menu de la Fig.82 dans les onglets bleu pastel. En cliquant sur l'onglet **Wiew Key** de ce menu on fait afficher une fenêtre qui ressemble à celle de la Fig.95 qui résume les paramètres de l'initialisation virtuelle adoptée sur la machine. Dans la zone repérée en jaune est précisé le nom du fichier de type **eni**



chargé, ainsi que son chemin sur la mémoire de masse de l'ordinateur. **ATTENTION, il faudra à chaque début de cryptage ou de décryptage commencer par replacer manuellement les orientations des Rotors en face de la fenêtre du plateau** car cette combinaison n'est pas mémorisée. Ce choix peut se justifier par le fait que chaque message reçu contient ce "préambule" de trois lettres et change à chaque transmission en fonction de critères qui sont propres aux services concernés dans les armées.



Disposant d'une machine de référence fiable et correctement configurée, nous pouvons passer à la campagne d'essais visant à valider le **Brouilleur** avec sérieux. Dans ce but, nous allons repartir du démonstrateur précédent, mais simplifié dans **P11\_Test\_integral\_du\_Brouilleur.ino** en supprimant l'affichage des cheminements qui n'apporte plus vraiment d'informations nouvelles et a tendance à surcharger l'affichage. Par ailleurs, ce nouveau démonstrateur a subi une petite modification. En mode TEXTE les caractères sont regroupés par paquets de cinq lettres pour avoir un affichage analogue à celui engendré par la chiffreuse de référence [Énigma].

### ➤ Modification de la configuration des Rotors et du Réflecteur.

Nous savons que nous devons adopter une initialisation totalement conforme à celle adoptée sur l'[Énigma] de référence. Naturellement, on commence par téléverser **P11** dans l'ATmega328 et on active le **Moniteur** à 57600baud.

#### MANIPULATIONS : (Suite)

- 10) Le **Moniteur** est en attente de consignes dans sa fenêtre de saisies.
- 11) Frapper **"er"** pour modifier le **Brouilleur**.
- 12) Proposer dans l'ordre **'5', '4', '3', '6', '15', '21', 'x', 'h' et 'a'**.
- 13) Imposer le **Réflecteur** avec **'c'**.
- 14) Par précaution sauvegarder cette initialisation avec **'s'** suivi de **'o'**.
- 15) Provoquer un nouveau RESET

et vérifier que la machine est bien configurée comme sur la Fig.96 pour s'assurer d'une initialisation conforme à celle du test que l'on désire effectuer.

```
>>> CONFIGURATION <<<
ROTOR de Gauche = V   Index de Gauche = 6 Orientation = X
ROTOR du Centre = IV  Index du Centre = 15 Orientation = H
ROTOR de Droite = III Index de Droite = 21 Orientation = A
REFLECTEUR = C
```

Fig.96

Un peu comme avec "Les plans d'expériences par la méthode TAGUCHI" on a changé plusieurs paramètres pour optimiser le test. On a modifié la nature des **Rotors**, ainsi que leur position, leur **Indexation** interne, les orientations sous la fenêtre et mis en place le **Réflecteur C** au lieu du **B**. Bref, l'intégralité de l'initialisation est bien complètement "chamboulée".

#### MANIPULATIONS : (Suite)

- 16) Sur [Énigma] la configuration de la Fig.95 a été téléchargée.
- 17) Sur [Énigma] on a orienté les trois Rotors en **'X', 'H' et 'A'**.
- 18) Commande **"t"** pour passer en mode TEXTE.
- 19) Passer en CRYPTAGE avec le caractère **'&'**.
- 20) Tester avec **"aaaaaaaaaaaaaaaaaaaaa"**. (Penser à Copier/Coller.)
- 21) Enfin **'&'** pour sortir. (Ci-dessus 21 caractères pour ne pas que le Rotor du centre ne tourne.)
- 22) Sur [Énigma] proposer également **"aaaaaaaaaaaaaaaaaaaaa"**.

Sur les deux machines nous obtenons exactement un chiffrement identique. Sur la Fig.97 en **A** on trouve les résultats sur notre prototype alors qu'en **B** ceux sur la machine de référence. En rose et vert le préambule dont les trois premiers caractères constituent l'orientation initiale des rotors et en orange l'état final sur les deux chiffreuses.

```
Passage au mode CRYPTAGE.
XHA IWQ GWY OAD
MDMZQ ZBUJQ EQREI DJTUH C
>>> [X H V]
```

**A**

Fig.97

```
AAAAA AAAAA AAAAA AAAAA A
MDMZQ ZBUJQ EQREI DJTUH C
```

**B**



- 23) Pour compléter ce test recharger sur [Énigma] la configuration de la Fig.95 sans oublier de recalculer manuellement l'orientation des trois **Rotors**.
- 24) Revenir en CRYPTAGE avec le caractère **'&'** en **'X', 'H' et 'A'**.
- 25) Proposer la chaîne **"abcdefghijklmnpqrstu"**. (Penser à Copier/Coller.)
- 26) Sur [Énigma] imposer le même texte.

On vérifie que quel que soient les rotors insérés et leur **Indexation** interne, avec le **Réflecteur B** ou **C** et des orientations initiales quelconques, quel que soit le caractère proposé en

cryptage, on obtient exactement des codages identiques. On peut donc valider avec sérénité l'ensemble du comportement du **Brouilleur** et envisager avec confiance la suite du développement.

### MANIPULATIONS : (Suite)

- 27) Sur notre prototype faire un RESET.
- 28) Revenir à la configuration de base de la Fig.83 : '1', '2', '3', '1', '1', '1', 'a', 'a', 'a' et 'b'.
- 29) Option "t" pour confirmer le mode TEXTE.
- 30) Consigne '&' pour retourner en CRYPTAGE.
- 31) Proposer "**alooa woqlw neict ottyi lvalg koacs acoyd exxkd gnsno gmior**".
- 32) Puis "**gfzzn udnud vctsq xzeth bcbh sfmyo hilnf ykpne nmsd**".

**Note :** Le message complet a été fractionné en deux pour rester dans la marge de 60 caractères maximum limités par le **Moniteur de l'IDE**. Il suffit de placer autant de "morceaux" de textes dans la fenêtre de saisie sans sortir du cryptage avec '&'. Il y a continuité du listage correctement formaté. On peut par cette technique transmettre des messages aussi longs que désiré.

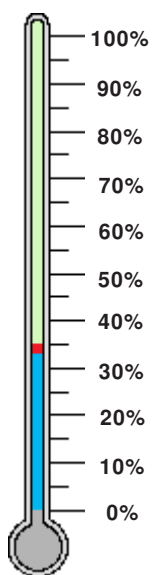
```
Passage au mode CRYPTAGE.  
AAA IWQ GWY OAD  
BENMO AMOAJ ETROU VEINC OHERE NTDEN OMMER CHIFF REUSE DANSL ETUTO RIELU  
NEMAC HINEQ UINEF OURNI TOUED ESLET TRES  
>>> [A A Q]  
Retour au mode COMMANDES.  
COMMANDE ->
```

Fig.98

Un tel texte n'est pas évident à lire. Il importe donc de le reprendre en reconstituant les mots et en y ajoutant les espaces. On peut alors obtenir en clair la pensée philosophique de la petite salamandre exprimée sur la Fig.90 en mode crypté : **BEN MOAMOA JE TROUVE INCOHERENT DE NOMMER CHIFFREUSE DANS LE TUTORIAL UNE MACHINE QUI NE FOURNIT QUE DES LETTRES !** C'est la raison pour laquelle plusieurs protocoles étaient prévus pour transmettre des noms propres, intercaler "des ponctuations" etc. Les chiffres et les nombres étaient exprimés en toutes lettres et tout un travail de présentation était effectué par les opérateurs radio de l'époque.

Fig.99

### ➤ Nouveau bilan sur l'évolution du programme.

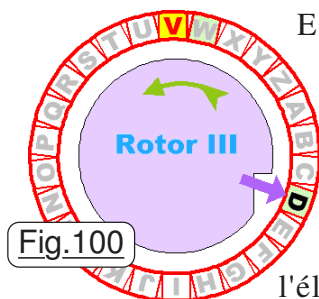


Depuis la dernière évaluation sur la Fig.81 en page 45 nous avons intégré des routines relativement complexes et maintenant le **Brouilleur** est entièrement opérationnel et fiable. Pourtant, en comparaison avec le dernier rapport de ce type, la situation en termes de taille occupée par le programme n'a pas changé de façon considérable. S'il est vrai que certaines optimisations ont été ajoutées diminuant la consommation des octets, des séquences relativement complexes ont été ajoutées. Pourtant, l'occupation de la mémoire réservée au programme dépasse à peine le tiers de l'espace dédié. On peut affirmer sans grand risque de se tromper que lorsque le programme sera achevé, on sera probablement loin de la saturation. Aussi on peut envisager l'avenir avec légèreté ... mais pour le principe nous continuerons à optimiser au maximum le code. Si on tente dans l'état actuel de proposer la commande 'p' au démonstrateur **P11**, la zone libre entre la **PILE** et le **TAS** avoisine 1304 octets, soit un espace largement suffisant pour loger les données "fugitives". Quand à l'espace réservé aux variables globales, seul 35% est actuellement consommé, laissant plus qu'il n'en faut de place pour les variable dynamiques. Donc pour le moment ... tout va bien !

### 21) Le mécanisme fictif d'entrainement des Rotors.

Loin d'être élémentaire, le mécanisme qui sur les machine réelle générant l'incrémentation des **Rotors** risque fort de nous donner pas mal de "fil à retordre". En effet, ce n'est pas du tout aussi simple qu'un compteur numérique en base 26 et ce pour plusieurs raisons. La première réside dans le fait que pour chaque **Rotor** installé sur la machine, il engendre le mouvement pour la roue voisine à sa gauche pour des transitions différentes. Et encore, ne nous plaignons pas. Pour la machine émulée dans ce didacticiel, les **Rotors** n'engendrent qu'une seule incrémentation par tour contrairement aux extensions disponibles sur d'autres versions. Le tableau de la Fig.100 précise pour les cinq éléments de notre "magasin" les lettres d'incrémentations sur la bague.

Rotor	Encoche	Fenêtre	Transition
I	Y	Q	Q > R
II	M	E	E > F
III	D	V	V > W
IV	R	J	J > K
V	H	Z	Z > A



Exemple pour le **Rotor III** :  
 Quand l'encoche d'entraînement est en **D**, c'est la lettre **V** de la bague extérieure qui est visible sous la fenêtre de la machine. Lorsque ce **Rotor III** passe de **V** à **W** il entraîne en rotation l'élément situé à sa gauche.

Autre exemple : Supposons que le **Rotor I** est au centre. Son encoche est en **Y**. Si cette encoche est positionnée devant le cliquet, la lettre **Q** est visible dans la petite fenêtre. Par conséquent, le rotor à gauche du **Rotor I** (*Donc celui tout à gauche du Brouilleur.*) avancera si ce dernier passe de **Q** à **R**.

### ➤ Le double pas.

L'ensemble des trois **Rotors** semble fonctionner comme un "compteur kilométrique" avec des rouages qui présenteraient 26 positions par cycle, et l'élément des unités (*Le rotor le plus à droite*) qui s'indexerait à chaque touche activée sur le clavier. La roue du centre serait incrémentée chaque fois que les "unités" effectuent un tour complet. Puis celle de gauche chaque fois que le **Rotor** du centre boucle à son tour un cycle de 26 lettres. Mais il y a une différence importante résultant du mécanisme des cliquets et de la denture. Le **Rotor** du milieu avance une deuxième fois de suite sur le pas suivant du **Rotor** de droite, si le **Rotor** du centre est dans sa propre position de rotation. *C'est ce que l'on nomme le double pas.*

Exemple d'une telle séquence lorsque les éléments **I - II - III** sont installés dans cet ordre :

**KDT** : Orientations initiales sur les trois **Rotors**.

**KDU** : Rotation simple du **Rotor** de droite. (*Nommé aussi le Rotor rapide.*)

**KDV** : Rotation simple du **Rotor** de droite.

**KEW** : Entraînement simple à gauche mais "arme" en **E** l'encoche de la roue du centre.

**LFX** : **Deuxième pas au centre** car l'encoche était armée.

**LFY** : Rotation simple du **Rotor** de droite.

**LFZ...** : Rotation simple du **Rotor** de droite ...

Comme on le constate, passer de **V** à **W** engendre la rotation de l'élément central. Hors ce mouvement place un **E** sous la fenêtre. Donc son encoche **M** est alors "armée". Du coup, lettre suivante quand le rotor rapide passe de **W** à **X**, le **Rotor** central est à nouveau sollicité et passe de **E** à **F**. C'est la conception mécanique des cliquets et des dents le loup qui engendre ce comportement assez singulier nommé **double pas**. Ce type de mécanisme est utilisé sur l'Énigma de la Wehrmacht et de la Kriegsmarine (*Donc celle que nous allons simuler.*) mais pas sur le modèle G.

### RESUMÉ :

- Le Rotor de droite tourne à chaque caractère.
- Lorsque la bague transite de la "lettre encochée" à la suivante, le **Rotor** central s'incrémente.
- Si par cette incrémentation le **Rotor** du centre présente alors sa "lettre encochée", **alors le caractère suivant** engendrera sa double incrémentation.

### ➤ Deux petits plus pour agrémenter l'ordinaire.

C'est **P12\_Mouvement\_des\_trois\_Rotors.ino** qui va se charger d'introduire dans la structure immatérielle de notre codeuse les mécanismes qui animent en rotation les trois **Rotors**. On se doute que cette cinématique virtuelle intégrera le "Double pas", thème de ce chapitre. Lors de cette nouvelle modification du logiciel, deux options secondaires se sont avérées bien utiles et ont été émulées dans **P12**. Ce sont deux nouvelles commandes "luxueuses" et on peut parfaitement s'en passer. Toutefois, comme les évaluations actuelles supposent que l'on va disposer de la place à revendre, on peut se permettre de dilapider notre richesse, quitte plus tard à enlever ces petites améliorations si le besoin s'en fait sentir. La première option concerne surtout les programmeurs, raison pour laquelle elle est suspendue par défaut. Elle permet comme montré sur la Fig.101 de préciser en fonction des **Rotors** installés dans le **Brouilleur** leurs "lettres encochées" qui engendrent les mouvements à gauche de l'élément considéré.



```
>>> CONFIGURATION <<<
ROTOR de Gauche = V Index de Gauche = 6 Orientation = X
ROTOR du Centre = IV Index du Centre = 15 Orientation = H
ROTOR de Droite = III Index de Droite = 21 Orientation = A
REFLECTEUR = C Encochage[ZJV]
FICHE n°01 -> [B et J] FICHE n°02 -> [C et Q]
FICHE n°03 -> [D et I] FICHE n°04 -> [F et M]
FICHE n°05 -> [G et O] FICHE n°06 -> [H et R]
FICHE n°07 -> [K et W] FICHE n°08 -> [P et Y]
FICHE n°09 -> [T et U] FICHE n°10 -> [X et Z]
GROUPE d'identification = XHA IWQ GWY OAD
```

Fig.101

le **Brouilleur**. L'affichage de l'Encochage indique les "lettres d'incrémentation" qui sont indépendantes de l'Indexation interne des **Rotors** ou de leur orientation au cours de l'utilisation de la machine. Ces trois lettres sont à comparer avec celles pour **V, IV et III** du tableau de la Fig.100 proposé en page précédente. L'autre option qui a été ajoutée au **Menu de base** est déclenchée avec la nouvelle commande '**d**'. Elle permet d'imposer à Énigma l'**Initialisation par défaut** du simulateur [Énigma] que l'on utilise comme référence et qui a déjà été plusieurs fois installé sur la machine.

## MANIPULATIONS :

- 01) Téléverser **P12** sur la carte Arduino NANO et activer le **Moniteur** de l'**IDE** à 57600 baud. Noter dans le **Menu de base** les deux nouvelles commandes '\*' et '**d**'. *Remarquer au passage que les commandes sont maintenant listées dans l'ordre alphabétique ce qui est plus rationnel.*
- 02) Frapper '\*' pour valider l'affichage de l'**Encochage**.
- 03) Proposer '**c**' pour faire afficher la configuration de notre prototype. On peut vérifier que les informations d'encochage sont bien **[ZJV]** conformément aux données attendues.
- 04) Répéter '\*' pour suspendre cette nouvelle information.
- 05) Recommencer avec '**c**' pour vérifier que cette option est à nouveau suspendue.
- 06) Réhabiliter la visualisation de l'**Encochage**, toujours avec la commande '\*'.
- 07) Commande '**d**' suivi de '**x**'. Il ne se passe rien, car la réponse n'étant pas un '**o**' est considérée comme une NON confirmation.
- 08) On recommence avec '**d**' confirmé cette fois par '**o**'. Conformément à nos attentes, la configuration actuelle est bien celle par défaut d'[Énigma] de référence.
- 09) Enchaîner les deux consignes '\*' puis '**c**' : L'encochage affiché est bien **[QEV]** correspondant à la nouvelle combinaison des **Rotors** installés sur le **Brouilleur**.

### ► Une troisième option en promotion.

**C** érifier le comportement de l'encliquetage virtuel impose de nombreuses manipulations à partir d'une configuration initiale. L'expérience montre qu'il faut souvent ne modifier que l'orientation des trois **Rotors** sans changer le reste de la configuration. Aussi, "j'ai craqué" et ajouté dans le **Menu de base** l'option '**a**' pour **Arranger les Rotors**. Cette commodité ne consomme que 112 octets dans la mémoire réservée au programme et 20 emplacements en mémoire dynamique. Il reste encore 1248 octets entre la **PILE** et le **TAS**. À ce prix il n'y a vraiment aucune raison de s'en priver.

## MANIPULATIONS : (Suite)

- 10) Frapper la commande '**a**' pour ouvrir la séquence d'orientation des trois **Rotors**.
- 11) Indiquer '**s**', puis '**k**' puis '**f**' par exemple.
- 12) Enfin, avec '**c**' faire lister la nouvelle configuration. Seules les trois orientations ont changé, ainsi que les trois premières lettres du **GROUPE d'identification**. Conviviale cette option ...



Dans la page précédente Nulentuk précise qu'il va y avoir de la place à revendre dans le microprocesseurtrucmachin. C'est idiot, car on ne peut pas vendre des cellules élémentaires de la mémoire réservée au programme !

➤ **Vérifier la rotation des deux rotors de droite avec influence du Double pas.**

Commencer par les deux premiers **Rotors** est une approche à la fois simplifiée et historique. Simplifier est une évidence, car pour engendrer la rotation du **Rotor** de gauche il faut frapper 26 x 26 soit 676 lettres sur le clavier virtuel de la machine. Historique, car pour diverses raisons de sécurité des transmissions secrètes, les opérateurs manipulant Enigma avaient pour consigne d'éviter les textes trop longs et scinder les transmissions longues en plusieurs messages courts, chacun indépendant avec son groupe d'identification et son préambule associé.

**MANIPULATIONS :** (Suite)

• **Commençons par un test simple sans Double pas. Le logiciel est en mode COMMANDE.**

- 13) On impose "er" pour imposer une initialisation quelconque.
- 14) Puis frapper dans l'ordre '3', '5', '2', '1', '1', '1', 'p', 'n', 'c' et 'b' par exemple.
- 15) Par précaution : 's' suivi de 'o' pour sauvegarder en EEPROM.
- 16) Passer en CRYPTAGE avec '&'.
- 17) Proposer la suite 'w', 'w', 'w', 'w', 'w' pour réaliser le test. (1)

**PNC** : Initialisations de départ sur les trois **Rotors**.  
**PND** : Rotation banale du **Rotor** de droite. (Nommé aussi le *Rotor rapide*.)  
**PNE** : Rotation simple du **Rotor** de droite.  
**POF** : Entrainement en **O** à gauche du **Rotor** de droite qui passe de **E** à **F**.  
**POG** : Rotation suivante du **Rotor** de droite.  
**POH** : Mouvement suivant du **Rotor** de droite ...

- 18) Conditionner de façon identique l'[**Énigma**] de référence. Effectuer la manipulation analogue pour vérifier que les deux machines fournissent des résultats identiques.

• **Effectuons maintenant un test avec intervention du Double pas.**

- 19) Revenir au mode COMMANDE avec '&'.
- 20) Proposer 'a' suivi de 'g', 'y', 'c' pour placer la machine en "configuration de Double pas".
- 21) Reprendre le CRYPTAGE avec '&'.
- 22) Pour le message indiquer 'w', 'w', 'w', 'w', 'w'.

**GYC** : Configuration de départ sur les trois **Rotors**.  
**GYD** : Rotation ordinaire du **Rotor** rapide.  
**GYE** : Mouvement suivant du **Rotor** de droite.  
**GZF** : Passage en **Z** à gauche du **Rotor** de droite qui franchit **E** vers **F**.  
**HAG** : **Deuxième pas au centre** car son encoche **Z** était armée. Mais comme le rotor central transite de **Z** vers **A** il entraîne le **Rotor** à sa gauche de **G** vers **H**.  
**HAH** : Mouvement suivant du **Rotor** de droite ...

- 23) Il est incontournable d'effectuer des manipulations identiques sur l'[**Énigma**] de référence.

• **Réalisons un test sans le Double pas avec mouvement du Rotor de gauche.**

- 24) Sortir du CRYPTAGE avec '&'.
- 25) Proposer 'a' suivi de 'n', 'z', 'b' pour cette vérification.
- 26) Retour en CRYPTAGE avec '&'.
- 27) Suite 'w', 'w', 'w', 'w', 'w' pour le jeu d'essais.

**NZB** : Configuration de départ avec Encoche **Z** sur le **Rotors** du centre.  
**OAC** : Entrainement direct du **Rotors** du centre et comme il transite de **Z** vers **A** il fait tourner également le **Rotor** à sa gauche.  
**OAD** : Rotation ordinaire du **Rotor** rapide.  
**OAE** : Mouvement banal du **Rotor** de droite.  
**OBF** : Passage en **B** à gauche du **Rotor** de droite qui franchit **E** vers **F**.  
**OBG** : Mouvement suivant du **Rotor** de droite ...

- 28) On se doute qu'il faut comparer ce comportement avec celui de l'[**Énigma**] de référence dont on se contente de recalculer manuellement l'orientation des trois **Rotors**.

(1) : Touche 'w' car peu utilisée en Français pour "économiser le clavier".

## MANIPULATIONS : (Suite)

### • Autre vérification qui engendre le mouvement des trois rotors à la première lettre frappée.

- 29) Sortir du CRYPTAGE avec '&'.
- 30) Proposer 'a' suivi de 'g', 'z', 'a' par exemple.
- 31) Revenir en CRYPTAGE avec '&'.
- 32) Suite avec 'w', 'w', 'w', 'w', 'w', 'w' pour le jeu d'essais.

**GZA** : Configuration de départ avec Encoche **Z** sur le **Rotors** du centre.  
**HAB** : Entraînement direct du **Rotors** du centre et comme il transite de **Z** vers **A** il fait tourner également en **H** le **Rotor** à sa gauche.  
**HAC** : Rotation ordinaire du **Rotor** rapide.  
**HAD** : Rotation classique du **Rotor** rapide.  
**HAE** : Mouvement banal du **Rotor** de droite.  
**HBF** : Passage en **B** à gauche du **Rotor** de droite qui franchit **E** vers **F**.  
**HBG** : Mouvement suivant du **Rotor** de droite ...

- 33) Comme à chaque fois on compare ce comportement avec celui de l'[Énigma] de référence.

### • Tester avec V à droite car il transite de Z vers A donc passe "de la fin au début".

- 34) Retour au mode COMMANDE avec '&'.
- 35) Frapper "er" pour imposer une nouvelle initialisation.
- 36) Proposer dans l'ordre '4', '3', '5', '1', '1', '1', 'f', 'j', 'x' et 'c' par exemple.
- 37) Pour la forme '\*' suivi de 'c', et vérifier [JVZ].
- 38) Par précaution : 's' suivi de 'o' pour sauvegarder en EEPROM.
- 39) On va chiffrer avec '&'.
- 40) Message abscons 'w', 'w', 'w', 'w', 'w', 'w' pour tester.

**FJX** : Configuration de départ avec Encoche **J** sur le **Rotors** du centre.  
**FJY** : Rotation ordinaire du **Rotor** rapide.  
**FJZ** : Rotation ordinaire du **Rotor** rapide.  
**FKA** : Passage en **K** sur le **Rotor** du centre car le rapide transite de **Z** vers **A**.  
**FKB** : Mouvement suivant du **Rotor** de droite.  
**FKC** : Mouvements banals du **Rotor** de droite ...

- 41) Pour terminer cet essai on compare avec [Énigma] de référence que l'on initialise entièrement.  
(N'oubliez pas de changer le réflecteur et de prendre le **C**)

### • Tester avec V à droite et intervention du Double pas.

- 42) Sortie avec '&' suivi de 'a' associé à 'f', 'u', 'x'.
- 43) On recommence les consignes '&', 'w', 'w', 'w', 'w', 'w', 'w' pour expérimenter.

**FUX** : Configuration de départ sur les trois **Rotors**.  
**FUY** : Rotation ordinaire du **Rotor** rapide.  
**FUZ** : Mouvement suivant du **Rotor** de droite.  
**FVA** : Passage en **V** à gauche du **Rotor** de droite qui franchit **Z** vers **A**.  
**GWB** : **Deuxième pas au centre** car son encoche **V** était armée. Mais comme le rotor central transite de **V** vers **W** il entraîne le **Rotor** à sa gauche de **F** vers **G**.  
**GWC** : Mouvements suivants du **Rotor** de droite ...

- 44) Incontournable comparaison avec [Énigma] de référence dont on a réorienté les **Rotors**.

### • On réitère la combinaison des trois rotations sur la première lettre frappée.

- 45) Touche '&' pour le mode COMMANDE suivi de 'a' associé à 'q', 'v', 'x'.

**PVX** : Configuration de départ avec Encoche **V** sur le **Rotors** du centre.  
**QWY** : Entraînement direct du **Rotors** du centre et comme il transite de **V** vers **W** il fait tourner "naturellement" son **Rotor** à sa gauche.  
**QWZ** : Mouvement banal du **Rotor** de droite.  
**QXA** : Passage en **X** à gauche du **Rotor** de droite qui franchit **Z** vers **A**.  
**QXB** : Mouvements suivants du **Rotor** de droite ...

- 46) Et l'inévitable comparaison avec [Énigma] dont on a remplacé les **Rotors**.



## 22) Tester des messages longs pour vérifier le comportement du Brouilleur.

C'est durant cette campagne de tests que l'on va transgresser les consignes qui étaient imposées aux opérateurs radio devant fractionner les messages pour ne pas qu'ils soient trop longs. Nous allons réaliser des transmissions fictives très longues et nous assurer que le démonstrateur fournit des résultats fiables par comparaison avec ceux de l'[Énigma] de référence. Pour ce dernier test du **Brouilleur** on va modifier tous les paramètres sur les deux machines, y compris les **Indexations** internes des **Rotors**. Les textes envoyés comporteront toutes les lettres de l'alphabet placées dans un ordre quelconque.

### ➤ Un jeu d'essai vraiment complet.

Comme pour notre premier essai on va soumettre un texte de 360 lettres, qu'il n'est pas question de frapper au clavier une à une et de valider à chaque fois. Nous allons donc utiliser un protocole très facile par usage du **Copier/Coller** dans la procédure qui va suivre. Dans ce but nous allons utiliser le message particulier de la Fig.102 dans lequel les 360 lettres sont ordonnées de façon aléatoire et repérées par groupes de dix. Chaque ligne comporte soixante caractères pour respecter la limite de saisie du **Moniteur de l'IDE**. Mis à part les quatre dernières lettres, pour le reste, l'ensemble comporte le même nombre pour chaque lettre de l'alphabet.

abcdefghijklmnopqrstuvwxyzneogrdqpklazwmbvcfihjsxutyaqwzxsxed  
crfvtgbyhnujikoipmpoiuytrezaqsdfghjklmnbvcxwazertyuiopmlkjhg  
fdsqwxcvbnmlpkonjibhuvgyctfxdrwseqzawqaxszcdevfrbgtnhyjukilo  
mpaqzsedrftgyhujikolpmnwbxvcwqxsdvfbgnhkjmlpouiyterzaawzxec  
rvtbynukjiomlpqdfsgghazqersdtyfguihjopklmwxvxbnqazsderfgythju  
awzxecrvtbynsqfdhgkjmluipoplokjihuygtfdrsezqawvxbcnaxtnipjfs

Fig.102

### MANIPULATIONS :

- 01) Pour la forme "repartir de zéro" avec un RESET.
- 02) Éventuellement frapper un 'm' pour valider le bruiteur.
- 03) Frapper un 't' pour imposer le mode TEXTE.
- 04) On prend le manipulateur Morse virtuel avec '&'.
- 05) **Copier** dans la Fig.102 la ligne du haut.
- 06) Dans la fenêtre saisie du **Moniteur Coller** ce texte et **valider**.
- 07) **Copier** dans la Fig.102 la ligne avec les lettres oranges.
- 08) **Coller** ce texte dans la fenêtre de saisie du **Moniteur** et **valider**. (Tant que l'on ne quittera pas le mode CRYPTAGE avec '&' le démonstrateur continuera à chiffrer et présenter proprement.)
- 09) Continuer ces **Copier / Coller** avec les lignes à caractères violets, marron, kaki et bleus.
- 10) Revenir en mode commande avec la consigne '&'. On obtient le résultat de la Fig.103 dans lequel les trois premières lettres du **GROUPE d'identification** sont repérées en rose et le trio final sous la fenêtre de la machine **[EFS]** repéré en vert pastel.

```

Passage au mode CRYPTAGE.
ERW IWQ GWY OAD
OZSZC WRSEE WUGZW DJMRF QCOIM EFRPS AZGXF XIXDA VNXZO DTXZY IVUZV IBPFC
DLBYI CLIYY GPNZU ALTWU XPFJI DARP NHLOG WNNIR QHZSR OTFGE AOCXW XSRFQ
DMPMF ASRHT XSVYT FFUQA CUBBQ RFCBG UWSOD PZXZU XBLWH BTMYQ ZXJLB LQWJR
VGBUN PCFMU JQLER RSPRK HCRKC OULFO FDQWE CLLTK AAQKG LJLGC OLHPX DDUXQ
DXEJP VQGTD BGZIR MKPYB RLGHN DNNFT CMCAC ZXJRE RMCOJ AGHRV UQXEW QJBNY
RNWZB TDUAW MAMPG HAEXN JIXWD CNYSE WJATD YETLM EDBTQ OIMGR AJOJM PCPOT
>>> [E F S]
Retour au mode COMMANDES.

```

Fig.103

- 11) Sur l'[Énigma] de référence on vérifie qu'en bas du coffret la petite fenêtre de visualisation des textes soumis et chiffrés est ouverte. Dans son menu de la Fig.82 cliquer sur **View Key**.
- 12) Configuration de la codeuse vérifiée orienter ses **Rotors** en **[ERW]**.
- 13) Dans le menu valider maintenant l'option **Auto Typing**.
- 14) Dans la Fig.102 **Copier** l'intégralité du texte.
- 15) Dans la grande fenêtre de saisie supposée vide **Coller** l'intégralité du texte. Choisir dans la petite fenêtre d'option en bas à gauche la rapidité **Very Fast**.

- 16) Terminer par **Start**. Le programme enchaîne les manipulations à une cadence impressionnante et termine comme sur notre prototype sur la combinaison **[EFS]** ce qui est de très bonne augure.
- 17) Un dernier clic dans la petite fenêtre d'affichage des deux lignes en bas de la machine. Une nouvelle fenêtre contextuelle s'ouvre, et à l'intérieur on peut observer que l'intégralité des 360 caractères chiffrés correspond bien à celle formatée par **P12**.



Ce test sévère nous permet de considérer que le **Brouilleur** est totalement fiable.

### ➤ **Brouillons les pistes !**

**F**açon comme une autre d'indiquer que par ce test ultime du **Brouilleur** nous allons encore modifier tous les paramètres sur les deux machines "en concurrence". Pour le plaisir, cette dernière expérience se fait "en situation", c'est à dire que nous allons reprendre l'initialisation dans la table de la Fig.47 mais cette fois pour le **24 Février 1942** sauf qu'ici le tableau des **FICHES croisées** ne sera pas pris en compte car non émulé pour le moment sur le démonstrateur **P12**.

### **MANIPULATIONS :** (Suite)

- 18) Reprendre "**er**" pour modifier uniquement le **Brouilleur**.
- 19) Frapper dans l'ordre '**3**', '**5**', '**2**', '**4**', '**17**', '**13**', '**j**', '**d**', '**h**' et '**b**' correspondant au 24/02/1942.
- 20) Consigner '**t**' puis '**&**' pour revenir en mode CRYPTAGE.
- 21) Pour cet exemple on va soumettre aux deux machines un très long texte de 1440 lettres. Dans ce but on va saisir sur le clavier virtuel quatre fois les 360 lettres de la Fig.102 ce qui impose on s'en doute une grande rigueur car on devra réitérer vingt quatre fois le **Copier/Coller** précédent.
- 22) Sortir du mode avec '**&**'. (*OUF !*) La petite fenêtre de la machine affiche **[LIR]**.
- 23) Sur **[Énigma]** de référence c'est bien plus convivial. On reprend exactement la procédure précédente en débutant par l'initialisation rigoureuse de la chiffreuse.
- 24) Réitérer l'option **View Key** pour s'assurer de la bonne configuration.
- 25) Ne pas oublier d'orienter les trois **Rotors** dans la combinaison **[JDH]**.
- 26) Redonner le long texte à **Auto Typing** en effectuant cette fois quatre **Copier/Coller** du texte complet. **ATTENTION** : Le premier coller doit écraser le texte initial.
- 27) Valider avec **Start**. C'est ici que l'on apprécie la rapidité du traitement.
- 28) Refaire strictement la même manipulation mais en donnant pour orientations de départ la combinaison particulière **[VZY]** qui est prévue pour engendrer dès la première lettre soumise à la machine l'entraînement des deux Rotors de gauche.

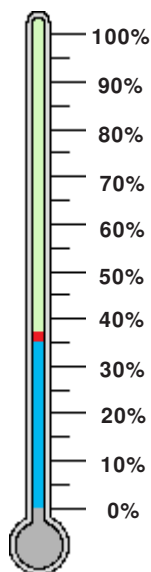
### • **Pour finir nous allons effectuer un test ultime en brouillant encore les cartes et 3000 lettres !**

Rassurez-vous, avec un tant sois peu de méthode ce sera un jeu d'enfant. Action :

- 29) On débute cette dernière expérience en réinitialisant les **Rotors** dans une nouvelle configuration inédite du genre '**3**', '**1**', '**4**', '**24**', '**9**', '**10**', '**l**', '**g**', '**y**' et '**c**'.
- 30) Puis les deux commandes classiques '**s**' confirmé par '**o**' pour le cas où nous désirerions effectuer encore quelques essais avec cette configuration.
- 31) Frapper '**t**' pour confirmer le mode TEXTE puis '**&**' pour débiter le CRYPTAGE.
- 32) **Copier** dans la Fig.102 la quatrième ligne avec les caractères marrons.
- 33) Faire cinquante fois **Coller/Valider** dans la ligne de saisie du Moniteur.
- 34) Revenir en mode COMMANDES avec '**&**' la fenêtre indique **[QWI]**.
- 35) Sur **[Énigma]** imposer la même configuration sans oublier de changer le **Rélecteur** en **C**.
- 36) Ne pas oublier d'orienter les **Rotors** en **[LGY]**.
- 37) Consigne **View Key** pour vérifier la conformité et invoquer **Auto Typing**.
- 38) Écraser le texte présent par dix fois **Coller**.
- 39) Copier le texte ainsi formé et le **Coller** quatre fois. (*Façon rapide d'effectuer 50 clones.*)
- 40) Valider : Festival de transmission Morse à cadence automatique et POFFffff ... on termine ce long chiffage avec une fenêtre qui affiche le fameux **[QWI]**. (*C'est bon pour le moral !*)
- 41) Un dernier petit clic de souris dans la petite fenêtre d'affichage des deux lignes en bas du coffret. La fenêtre contextuelle s'ouvre et affiche un contenu strictement identique à celui produit par notre prototype. Difficile d'effectuer un test aussi exigeant que celui-ci.

**CONCLUSION :** Les deux représentants de l'illustre codeuse présentent strictement des comportements identiques et sur trois derniers tests pour le moins sévères. Nous pouvons raisonnablement considérer que notre **Brouilleur** est totalement conforme à celui de la chiffreuse réelle et maintenant envisager d'émuler le "câblage immatériel" du tableau des **FICHES croisées**.

➤ **Un état des lieux en conclusion.**



**F**orce est de constater que pour ajouter la nouvelle fonction qui fait tourner le **Rotor** central et celui de gauche, nous n'avons consommé que 2% de la zone réservée au programme, l'occupation passant de 35% à 37%. Ayant également ajouté l'option de configuration par défaut, celle de modification des orientations initiales et la commande '\*', le dialogue Homme/Machine s'est enrichi. L'augmentation des textes affichés consomme 59 octets dans la mémoire dynamique dont l'occupation passe de 35% à 38% ce qui laisse encore 1254 octets de disponibles. Ce n'est certainement pas l'écriture des séquences pour simuler le tableau des fiches croisées qui va engendrer une boulimie d'octets, on peut donc penser sans risque de se tromper que nous n'atteindrons probablement pas 50% d'utilisation des ressources de l'ATmega328.

Conclusion n°1 : Il est tout à fait envisageable de réaliser une réplique matérielle de la codeuse ÉNIGMA avec un vrai clavier, un tableau de fiches croisées réel, des rotors moulés en 3D pour aboutir à un ensemble réaliste. Toutes les routines de traitement de l'information seront directement récupérables et surtout il restera probablement assez de place pour gérer des moteurs animant les roues du **Brouilleur**, un petit écran pour permettre l'initialisation, un bruiteur pour les erreurs etc. Optimiser comme nous l'avons fait engendre un gaspillage colossal, puisque sur la carte NANO nous n'allons consommer que la moitié des ressources. Consolons-nous ... ce n'est qu'un gaspillage immatériel. En revanche la place économisée ouvre des perspectives vraiment alléchantes. Une dernière petite conclusion me semble utile :

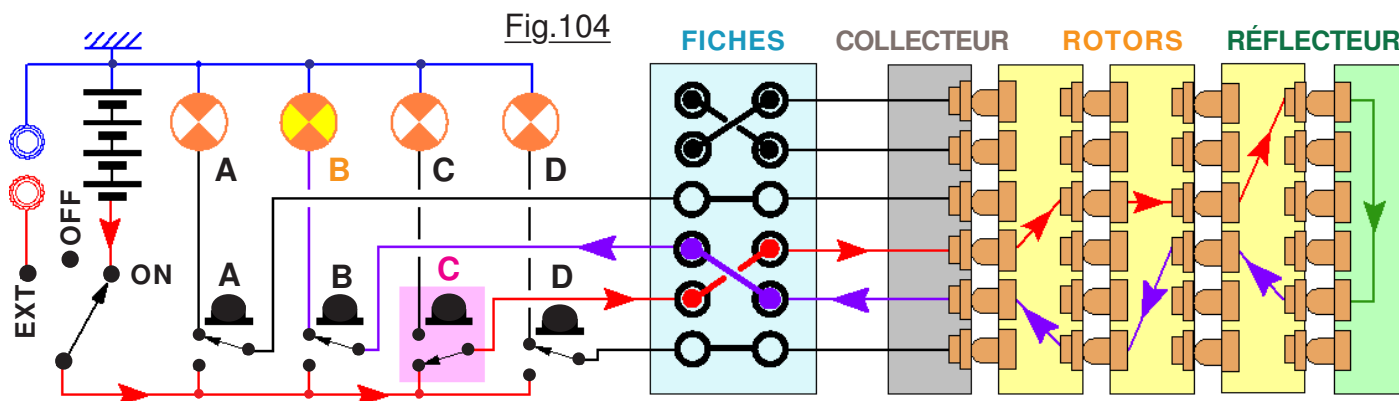
**STRATÉGIE de programmation.** *La lisibilité d'un programme est sa plus grande qualité.*

Pour gagner de la place nous avons logé un maximum de textes affichés dans la mémoire EEPROM. Cette technique à privilégier engendre toutefois une petite difficulté dans la lisibilité. En effet, comparer par exemple les deux instructions suivantes :

```
(1) void Aff_ROTOR() { ESPACE(); Aff_TEXTE_EEPROM(462,5); ESPACE(); }
(2) void Aff_ROTOR() { Serial.print(" ROTOR "); }
```

(1) est nettement moins facile à lire que (2).

**CONCLUSION :** Stocker un maximum de textes en EEPROM nuit à la lisibilité du programme. Aussi, en développement, commencer comme en (2) à logger tous les textes dans le programme. Ne songer à en passer en EEPROM que si un manque de place se fait sentir. Du coup, on saura exactement à ce stade la place disponible en EEPROM et on pourra optimiser l'opération en sélectionnant les textes les plus utilisés.



Sur la Fig.104 on frappe un **C** et la codeuse allume l'ampoule du **B** suite aux diverses permutations. Dans notre machine virtuelle actuelle nous disposons des **Rotors** et des **Réflexeurs**. Ils sont câblés virtuellement entre eux. Il ne reste plus qu'à effectuer le câblage logiciel du tableau des **FICHES croisées** ici en bleu clair et notre ÉNIGMA sera complète, objet du chapitre suivant.



### 23) Le tableau des Fiches Croisées.

**D**ernier module virtuel à ajouter à notre prototype logiciel pour que notre machine soit complète. Cet élément intervient dans le cryptage en entrée et en sortie du **Brouilleur**. Le programme va devoir en tenir compte. Les séquences impliquées dans le traitement du tableau des **FICHES croisées** devront fonctionner quel que soit le nombre de lignes électriques fictives installées sur la machine. Il reste toutefois à corriger un défaut dans la procédure d'initialisation des **FICHES croisées** car dans les démonstrateurs précédents on était obligé d'en installer obligatoirement dix sur la machine.

#### ➤ Initialisation du tableau des fiches.

**C**ontrairement à ce qui était prévu dans les démonstrateurs précédents, maintenant on pourra si on le désire installer un nombre quelconques de lignes croisées sur Énigma. Le protocole modifié dans **P13\_EXPLOITER\_Enigma.ino** est identique à celui de **P12** sauf que si on saisit '&' au lieu d'un couple de lettres, il y a sortie immédiate de la fonction et retour au mode COMMANDES.

#### MANIPULATIONS :

- 01) Téléverser **P13\_EXPLOITER\_Enigma.ino** et activer le **Moniteur** de l'**IDE** à 57600bauds.
- 02) Éventuellement frapper un '**b**' pour valider le bruiteur.
- 03) Frapper "**ef**" pour n'initialiser que les **FICHES croisées**. (Maintenant affiche **Fuite par '&'**.)
- 04) Proposer "**aq**", "**zs**", "**ed**", "**rf**" et '**&**' par exemple.
- 05) Indiquer '**s**' suivi de '**o**' pour sauvegarder.
- 06) Faire un **RESET** : Sauvegarde et restitution prennent bien en compte le nombre des Fiches.

**NOTE :** Pour ne transférer que le nombre de fiches installées en EEPROM il faut bien enregistrer dans cette dernière leur nombre. C'est ici que l'on est content qu'en EEPROM il reste un peu de place disponible. Du coup, l'octet situé en 669 contient cette information et il ne reste plus qu'un seul emplacement de libre en 668. Cette modification est commentée dans la **Fiche n°11**.

- 07) Saisir "**ef**" et sortir immédiatement par '**&**'. On constate que c'est un moyen d'effacer toutes les **FICHES croisées** avec seulement deux commandes.
- 08) Recommencer avec "**ef**" et ajouter "**tg**", "**yh**", "**uj**" puis '**&**' pour brancher trois lignes croisées.
- 09) Imposer la consigne '**d**' et la valider avec '**o**'. Le programme nous demande si l'on désire débrancher toutes les fiches. Refuser avec une lettre quelconque. Les branchements sur le tableau virtuel n'ont pas été changés.
- 10) Répéter '**d**' suivi de '**o**'. Cette fois accepter le débranchement de toutes les **FICHES croisées**. Cette dernière manipulation procure l'état par défaut de l'**[Énigma]** de référence.

#### ➤ Dilapidons notre richesse.

**A**yant estimé que nous n'allons pas consommer la moitié des ressources de l'ATmega328, on peut se permettre sans vergogne de jeter des octets par la fenêtre. Aussi, si vous observez le **Menu de base**, vous allez y découvrir une nouvelle commande '**f**'.

#### MANIPULATIONS : (Suite)

- 11) On recommence avec "**ef**" pour initialiser les **FICHES croisées**.
- 12) Remplir toutes les fiches pour vérifier au passage qu'après la n°10 on sort automatiquement.
- 13) Tester '**f**' suivi de '**n**' puis de '**c**' pour vérifier.
- 14) Reprendre '**f**' avec '**o**' puis encore '**c**' pour s'assurer de la cohérence du traitement.

Ces quelques manipulations nous ont montré les diverses procédures qui permettent de n'installer que quelques **FICHES croisées** ou de pouvoir toutes les débrancher. Durant la mise au point des nouvelles séquences, j'ai été amené à réinitialiser entièrement la machine avec la commande '**i**'. N'ayant pas pratiqué cette dernière depuis plusieurs semaines, j'avais complètement oublié que pour



Ben Mòa môa, pour tester l'Émanigtruc j'ai une idée que même le Nulentout il n'y a pas pensé. Je vais faire un long message qui contient la totalité des écritures de la Bible.

saisir le **GROUPE d'identification** on ne doit indiquer que neuf lettres, les trois premières étant réservées pour l'orientation initiale des rotors et déjà indiquées à la machine.

➤ **Petite amélioration en saisie du Groupe d'identification.**

Disposant d'une forte réserve de place en mémoire de programme, il était concevable d'améliorer de dialogue Homme/Machine en peaufinant les textes et ainsi fournir un rappel des trois premières lettres déjà saisies pour que l'opérateur puisse retrouver le fait que neuf caractères sont suffisants et pas les douze du **GROUPE d'identification** complet.

**MANIPULATIONS :** (Suite)

- 15) Imposer "eg" pour initialiser le **GROUPE d'identification**.

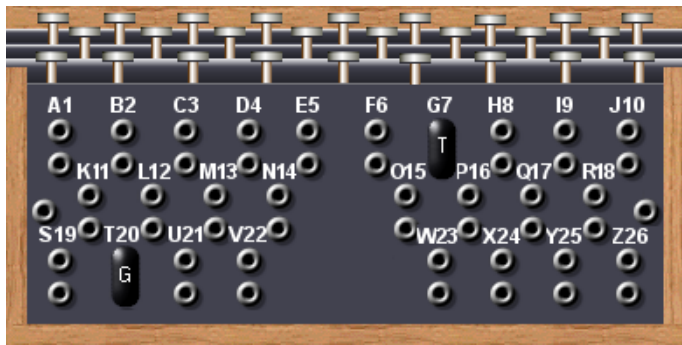
COMMANDE -> [EG] Fig.105  
 >>> Corriger la CONFIGURATION <<<  
 Compléter le GROUPE d'identification -> LGY[AAZZZEEE]

Avant le petit perfectionnement textuel, seule la zone en jaune sur la Fig.105 était affichée. Avec le logiciel ultime, le texte de la zone rose pastel précède l'ancienne invite et dans la surface repérée en vert un triplet redonne les trois premières lettre du **GROUPE d'identification** déjà disponibles.

- 16) Compléter avec neuf caractères comme "bouteille" par exemple. Le programme termine la ligne avec les lettres saisies placées entre crochets. (Sur la Fig.104 surlignées en violet.)

Avant de passer au chapitre suivant, il nous faut voir la procédure à utiliser sur l'[Énigma] de référence pour y installer des lignes croisées sur son tableau de Fiches.

- 17) Si elle est toujours active, quitter la machine "graphique" puis la réactiver avec sa configuration par défaut. Cliquer en **17** de la Fig.85 pour ouvrir le tableau des **FICHES croisées**.



Les vingt-six prises doubles sont réunies électriquement derrière le tableau par un petit pont électrique poussé mécaniquement par un ressort. *Si on n'insère pas de fiche croisée, les lettres sont toutes inchangées.* Pour installer virtuellement une ligne, il suffit de cliquer sur deux prises quelconques. Comme c'est le cas sur notre prototype, il est impossible d'utiliser deux fois la même prise. Si l'on clique sur l'une des

fiches déjà en place, on enlève purement et simplement le cordon considéré.

**MANIPULATIONS :** (Suite)

- 18) Pour établir une liaison électrique croisée sur cette chiffreuse virtuelle cliquer sur deux prises quelconques. Par exemple sur 'T' puis sur 'G'. Peu importe l'ordre dans lequel on installe la ligne.

*Pour ne pas cacher entièrement le tableau avec les fils de liaison électrique, ces derniers ne sont pas représentés. Seules les deux fiches sont affichées avec la "lettre croisée".*

- 19) Cliquer sur la fiche **G** en **T20** : La ligne croisée est immédiatement enlevée du tableau fictif.  
 20) Pour expérimenter, installer les trois lignes [EK], [ZW], [JP] et [DM] par exemple.

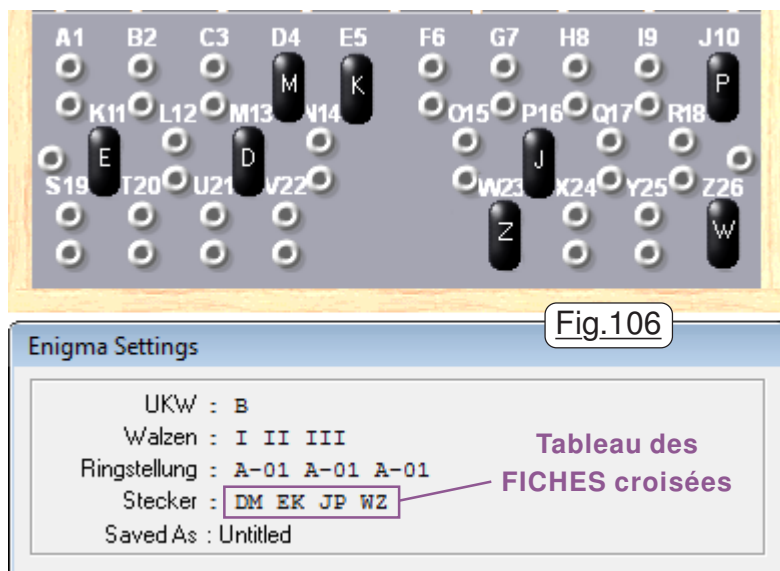
➤ **Tester le tableau virtuel des Fiches croisées.**

L'opération sera d'autant plus aisée que l'on pourra facilement ne brancher qu'une seule ligne si on le désire, tant sur le prototype que sur l'[Énigma] de référence. Comme pour les vérifications qui ont précédé, nous terminerons le jeu d'essai par un message vraiment très long précédé d'une configuration originale pour laquelle l'intégralité des paramètres possibles aura été modifiée. Ce message composé de caractères arrangés de façon aléatoire et en égale quantité pour chaque lettre de l'alphabet présentera une inflation coupable avec 6000 caractères dans son message final !



Dans la page qui précède, le Nulentout balance plein plein plein d'Octets par la fenêtre. C'est drôlement dangereux si en bas dans la rue un passant s'en prend un sur la tête !

Considérons le montage graphique de la Fig.106 avec mise en évidence de la mise en place actuelle des jonctions filaires sur le tableau des **FICHES croisées**. Elles ont été placées strictement dans l'ordre **[EK]**, **[ZW]**, **[JP]** et **[DM]**. Pourtant, quand on clique sur la commande **View Key** on constate sur la Fig.106 qu'elle ne sont pas du tout listées dans l'ordre de création comme c'est le cas sur notre prototype. À bien observer, et on peut le vérifier, elles sont classées par ordre alphabétique pour la première lettre.



### MANIPULATIONS : (Suite)

- 17) Faire un RESET sur le prototype façon de repartir de zéro.
- 18) Imposer la consigne 'd' et la valider avec 'o'. Puis encore 'o' pour débrancher toutes les fiches.
- 19) Lettres "ef" pour installer les quatre lignes déjà présentes sur l'[Énigma] de référence. Indiquer dans un ordre quelconque les quatre lignes virtuelles. Par exemple dans l'ordre "md", "wz", "pj", et pour finir "ke" et '&'. Les deux machines présentent à ce stade une initialisation commune.
- 20) Pour la forme on sauvegarde avec 's' confirmé par 'o'.
- 21) Nouvelle lettre 't' suivie de '&' pour retourner en CRYPTAGE.
- 22) Soumettre la chaîne "abcdefghijklmn" puis '&'.
- 23) Faire de même sur [Énigma] en mode **Auto Typing**. Les deux machines sont de connivence.

### ➤ De l'ordre nom d'une pipe !

Alphabétique pourrait compléter ce titre pour le moins impératif. Vous aurez certainement remarqué (*Pas forcément tout le monde !*) que dans le **Menu de base** figure la commande 'J' dont il n'a encore pas été question dans ce tutoriel. En fait, elle a été intégrée dans le programme **P13** tardivement et suite à la remarque sur la Fig.106 relative au triage des jonctions croisées. Du coup, comme je ne sais plus comment "faire enfler le programme" pour consommer des octets, cette petite option a été ajoutée à la liste. La commande 'j' liste les liaisons croisées, et au préalable les ordonne par ordre alphabétique comme sur la machine de référence. *Noter au passage, qu'à partir de cette évolution, quand on sauvegarde une configuration, les fiches sont systématiquement trillées avant de les mémoriser en EEPROM.* Naturellement, si au moment de la sauvegarde elles ont déjà été ordonnées avec 'j' il ne se passera rien de particulier.

- 24) Tester la commande 'j' qui, à l'instar d'[Énigma] liste les **FICHES croisées** ordonnées.

### ➤ Treize FICHES croisées au lieu de 10.

Potentiellement, sur une Énigma d'origine, rien n'interdit matériellement si son coffret contient un nombre suffisant de cordons électriques, d'utiliser toutes les prises du tableau des **FICHES croisées**. Sur une machine virtuelle comme la notre ou comme sur [Énigma] intégrer cette possibilité ne pose aucun problème et on ne va pas s'en priver avec toutefois une limitation. En effet, l'utilisation de l'EEPROM est actuellement limitée à dix fiches. En stocker trois de plus imposerait de revoir entièrement la gestion de la mémoire non volatile. On va donc se contenter d'élargir à 13 le nombre de **FICHES croisées** en exploitation, quitte à n'en sauvegarder que dix en EEPROM. Confirmation sur [Énigma] : Stecker : AB CF DM EK GH IR JP LS NV OX QY TU WZ .

### MANIPULATIONS : (Suite)

- 25) Provoquer un RESET sur le prototype façon de repartir de zéro.
- 26) Le classique "ef" pour réaffecter les croisements.
- 27) Indiquer dans l'ordre **[ab]**, **[fc]**, **[md]**, **[ek]**, **[hg]**, **[ri]**, **[jp]**, **[ls]**, **[vn]**, **[xo]**, **[qy]**, **[tu]**, et **[wz]** et terminer par 'j' pour trier tout ces fils.
- 28) Comme chaque fois, un petit 's' validé par 'o' pour sauvegarder en EEPROM.



- 29) Toujours en configuration par défaut imposer cette combinatoire à [Énigma].
- 30) Dans son menu commande cliquer sur **View Key** pour s'assurer de la conformité.
- 31) Sur notre prototype utiliser la commande 'j' pour comparer.
- 32) Nouvelle commande 't' suivie de '&' pour retourner en CRYPTAGE.
- 33) Soumettre la chaîne "abcdefghijklmnpqrstuvwxy" puis '&'.
- 34) Proposer ce texte à [Énigma] en mode **Auto Typing**. Les deux machines affichent exactement des cryptages identiques. On valide ainsi le prototype pour cette modification.

### > Compléter le tableau des FICHES croisées.

**R**estituer les dix **FICHES croisées** sauvegardées en EEPROM ne présente aucun intérêt si nous sommes obligés de tout reprendre avec 'i' ou "ef" pour retrouver une combinaison de plus de dix liaisons filaires. C'est la raison pour laquelle, et je suis certain que vous l'avez remarqué, le **Menu de base** a été complété par la commande 'n' qui autorise l'ajout de **Nouvelles** lignes filaires virtuelles sur le tableau d'Énigma. Fonctionne quel qu'en soit le nombre actuel. (*Sauf 13 !*)

### MANIPULATIONS : (Suite)

- 35) Faire un RESET sur notre machine pour récupérer les dix fiches sauvegardées.
- 36) Frapper 'j' pour faire afficher la liste ordonnées de celles qui ont été sauvegardées.
- 37) Il suffit de comparer les premières lettres qui sont dans l'ordre alphabétique pour déterminer facilement que ce sont les fiches [qy], [tu] et [wz] qui n'ont pas été enregistrées en EEPROM.
- 38) Frapper un 'n' pour ajouter des **Nouvelles** fiches. On constate que le programme affiche la liste des lettres déjà utilisées et continue bien à la Fiche n°11. Indiquer "qy" puis '&'. Dans cet exercice nous n'ajoutons qu'une seule fiche. Consigne 'c' pour vérifier. Recommencer un 'n'.
- 39) Proposer "tu" et "wz", arrivé à treize fiches il y a retour automatique au mode COMMANDE.
- 40) Terminer avec 'n' ... le logiciel refuse la commande car toutes les prises sont utilisées.

### > Enlever une FICHE croisée.

**A**vec la commande 'f' on débranche toutes les **FICHES croisées** en une seule commande. En enlever une seule peut sembler assez inutile. Et bien c'est pourtant une option très intéressante si par erreur sur une longue liste, (*En général dix dans les protocoles de l'époque.*) on constate que pour l'une d'entre elles nous nous sommes trompés. Supposons que dans l'action (27) au lieu de brancher [jp] il fallait employer [jw]. La procédure pour corriger ce type d'erreur consiste à enlever la ligne erronée avec 'k', (*'K' pour KILL car toutes les autres lettres significatives sont déjà attribuées.*) puis la remplacer avec usage de la commande 'n'. Expérimentons cette commande ajoutée 'k'.

### MANIPULATIONS : (Suite)

- 41) Déclencher un RESET, on va expérimenter cette nouvelle fonction qui vous l'aviez forcément remarqué est précisée dans le **Menu de base**.
- 42) Commande 'k' pour enlever une liaison croisée. Pour son numéro frapper un '&'. **Toute réponse différente d'une valeur numérique sera sans effet**. Pour sortir sans enlever de **FICHE croisée** on continuera à utiliser par habitude le '&' qui engendre toujours un changement de mode.
- 43) Reprendre 'k' suivi de "10". La dernière ligne filaire est enlevée.
- 44) Consigner 'k' avec "001". La première liaison est débranchée. **Les zéros en tête sont ignorés**.
- 45) Tenter 'k' complété par "9". La valeur dépasse le nombre actuel de fiches, le logiciel ignore la commande et précise à l'opérateur les nombres possibles.
- 46) Tester 'k' suivi de "a3". La commande est ignorée car commence par une lettre.
- 47) Frapper 'k' avec la réponse "3abcd". La consigne commençant par un nombre valide est acceptée, et les lettres qui suivent étant non significatives sont purement ignorées.
- 48) Lettre 'f' associée à 'o' pour enlever toutes les jonctions.
- 49) Proposer 'k' : Il y a sortie sans effet autre que nous prévenir de l'inutilité de cette tentative.
- 50) Encore un RESET pour retrouver dix fiches installées sur le tableau virtuel.
- 51) **Glups, la Fiche n°7 est erronée : [JP]**, alors que l'on désire [JW]. C'est ici que la possibilité d'enlever une seule ligne s'avère bien commode pour ne pas avoir à tout rebrancher. Frapper 'k' et désigner '7' pour enlever le mauvais cordon.
- 52) Proposer 'n' avec "jw" pour le remplacer ainsi que '&' et 'c' pour vérifier.

## ➤ La der des der(s) !

C'était promis au début du chapitre "*Tester le tableau virtuel des Fiches croisées*" en page 61, et l'on va terminer ces "hostilités" par un message un peu démentiel comportant 6000 caractères. Pour que ce test ultime soit le plus significatif possible, on va entièrement chantourloubouler la configuration initiale des deux machines mises en confrontation. Pour un maximum de "brassage" des lettres, on va entièrement remplir le tableau des **FICHES croisées**. Naturellement, nous allons également totalement modifier le **Brouilleur** dans une combinatoire nouvelle en changeant tous les paramètres. Vous l'avez deviné : Soumettre 6000 caractères en un seul message à nos deux chiffreuses impose méthode, discipline et surtout patience.

### MANIPULATIONS :

- 01) Comme à chaque étape importante, "repartir de zéro" avec un RESET.
- 02) Éventuellement frapper un '**b**' pour valider le bruiteur.
- 03) Imposer '**d**' et '**o**' pour commencer avec une configuration par défaut comme sur l'[**Énigma**].
- 04) Accepter de débrancher toutes les FICHES avec '**o**'.
- 05) Commande '**i**' qui n'a pas été beaucoup utilisée jusqu'à présent.
- 06) Frapper '**o**' pour confirmer et dans l'ordre '**2**', '**1**', '**4**', '**10**', '**26**', '**18**', '**k**', '**m**', '**b**' et '**c**'.
- 07) Pour le tableau des **FICHES croisées** indiquer dans l'ordre [**qa**], [**sz**], [**de**], [**fr**], [**gt**], [**hy**], [**ju**], [**ki**], [**lo**], [**mp**], [**xw**], [**vc**], et [**nb**] et terminer par '**j**' pour trier tout ces fils.
- 08) Pour le GROUPE d'identification indiquer "**bbbonjour**" par exemple. (*Neuf lettres au choix.*)
- 09) Consigner les deux complices '**s**' et '**o**'. (*On tombe dans la routine !*)
- 10) Configurer [**Énigma**] dans cette stricte combinatoire suivi de **View Key**.



Soit vous configurez en mode manuel, soit vous utilisez **load Key** du menu l'[**Énigma**] et adoptez **Der des der.eni** qui accompagne ce didacticiel dans le dossier <Documents>.

- 11) Commande '**c**' sur notre prototype pour vérifier encore les deux préparations.
- 12) Sur les deux machines tester quelques lettres : On doit obtenir des cryptages identiques.
- 13) Replacer sur les deux chiffreuses les rotors en [**kmb**].
- 14) Sur notre prototype en mode COMMANDE imposer '**&**' puis '**t**' et enfin '**&**' pour coder.
- 15) Pour soumettre les 6000 caractères à notre codeuse, on va coller 100 fois la chaîne de lettres "**rvtby nukjiomlpqdfsg hazqersdtyf guihjopklm wvxcbnqazsderfgtyhju**" puis sortir par '**&**'. Procéder par des [Copier/Coller] et comptez bien les occurrences.

Sur notre machine, [Copier/Coller] cent fois est assez rébarbatif, mais le résultat est immédiat.

- 16) Sur la machine de référence passer en mode **Auto Typing**. Commencer par Copier une fois le texte à crypter puis le Coller dix fois de suite. Copier dans la fenêtre de saisie ce bloc de six cent lettres et le coller encore neuf fois. Puis, la machine virtuelle étant toujours en option de rapidité **VeryFast** déclencher le codage avec **Start**. Aucun opérateur n'aurait cette cadence de frappe d'autant plus qu'à chaque lettre l'équipier doit noter le chiffrement sur du papier. On imagine assez bien le temps qu'il faudrait pour envoyer un tel message.

Bien que la cadence de frappe soit très rapide il faut beaucoup de temps à cette [**Énigma**] pour en arriver à bout. Soyez patient ... elle travaille pour nous et avec rigueur. Sur cette version du simulateur, rédiger le texte est bien plus rapide que sur la machine personnelle, mais pour crypter notre codeuse la surpasse très largement. Comme quoi chaque solution n'est qu'un compromis avec ses avantages et ses inconvénients. Considérons la Fig.107 qui présente les résultats obtenus sur les deux machines. Placées dans des conditions exactement identiques, sur un test particulièrement sévère, elles fournissent un codage strictement "superposable". Non seulement la vérification se fait avec un texte aléatoire de 6000 caractères présentant une distribution homogène des 26 lettres de l'alphabet, mais de plus le brouillage est maximal, tant par le choix des **Rotors** et de leur **Indexation** interne, complété par un tableau de **FICHES croisées** saturé avec treize lignes filaires installées. Aussi, *nous pouvons dormir sur nos deux oreilles*, le programme d'exploitation **P13** est largement validé et habilité à servir dans toutes les armées durant le conflit de 39/45. Au passage on peut se demander le temps qu'il faudrait pour coder un tel texte à la main si des machines comme **ÉNIGMA** n'existaient pas. On comprend assez bien le succès qu'à remporté cette chiffreuse automatique dans les armées, bien que vu sa complexité elle était très couteuse.

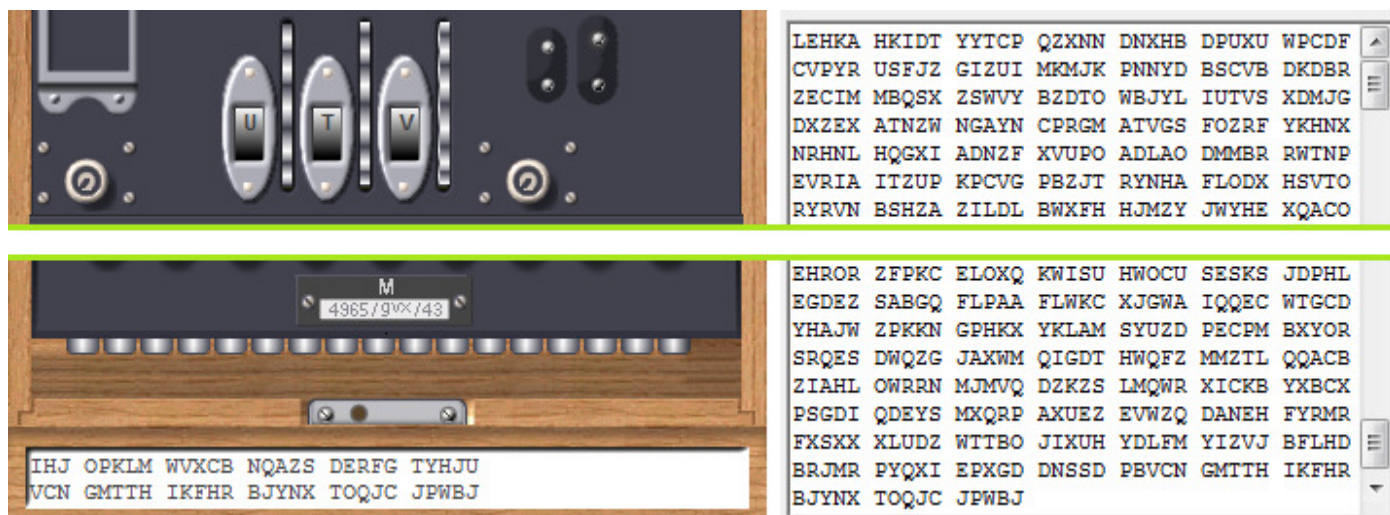


Fig.107

Confrontation des résultats obtenus sur les deux machines soumises à un texte commun de 600 lettres et placées dans des initialisations strictement identiques.

Passage au mode CRYPTAGE.  
 LEHKA HKIDT YYTCP QZXNN DNXHB DPUXU WPCDF CVPYR USFJZ GIZUI MKMJK PNNYD  
 BSCVB DKDBR ZECIM MBQSK ZSWVY BZDIO WBJYL IUTVS XDMJG DXZEX ATNZW NGAYN  
 CPRGM ATVGS FOZRF YKHNX NRHNL HQGXI ADNZF XVUPO ADLAO DMMBR RWTNP EVRIA  
 ITZUP KPCVG PBZJT RYNHA FLODX HSVTO RYRVN BSHZA ZILDL BWXFH HJMZY JWYHE  
 GNISV EHROR ZFPKC ELOXQ KWISU HWOCU SESKS JDPHL EGDEZ SABGQ FLPA A FLWKC  
 XJGWA IQQEC WTGCD YHAJW ZPKKN GPHKX YKLAM SYUZD PECPM BXYOR SRQES DWQZG  
 JAXWM QIGDT HWQFZ MMZTL QQACB ZIAHL OWRRN MJMVQ DZKZS LMQWR XICKB YXBCX  
 PSGDI QDEYS MXQRP AXUEZ EVWZQ DANEH FYRMR FXSXX XLUDZ WTTBO JIXUH YDLFM  
 YIZVJ BFLHD BRJMR PYQXI EPXGD DNSSD PBVCN GMTH IKFHR BJYNX TOQJC JPWBJ  
 >>> [U T V]

### ➤ Un dernier test pour vérifier la réversibilité de cette chiffreuse.

C'est le **Réfecteur** qui par conception confère à cette machine la faculté de crypter ou décoder sans rien avoir à changer sur la préparation matérielle. Une fois configurée pour la journée, l'opérateur radio pouvait aussi bien envoyer des messages qu'en recevoir. Rien n'est prévu de particulier dans **P13** dans ce sens, puisque c'est par "nature" qu'ÉNIGMA est "réciproque".

### MANIPULATIONS : (Suite)

- 17) Un petit RESET pour commencer. (Il aura servi le petit bouton bleu sur le coffret 3D.)
- 18) N'oublions pas le 't' car par défaut **P13** est en mode Lettre.
- 19) L'inévitable '&' pour commencer un cryptage.
- 20) Coder le message **"Bonjour les amis qui vont se faire une enigma."** (Copier/Coller.)
- 21) Revenir en mode COMMANDE avec '&' et **Copier** le texte dans la fenêtre du Moniteur.
- 22) Revenir en CRYPTAGE avec '&' et **Coller** le texte chiffré.
- 23) Dernier '&' pour revenir au **Menu de base**.

On notera au passage que les espaces et le point final ont été ignorés. Le texte est codé en zone verte. Puis soumis à nouveau à la machine il ressort en clair dans la zone rose, avec les limites imposées par le protocole, c'est à dire en groupes de cinq lettres. Notre ÉNIGMA est bien une machine qui fonctionne en chiffrement et en déchiffrement.

Passage au mode CRYPTAGE.  
 KMB BBB ONJ OUR  
 UYJOU LSAIJ DCBHM IQUPH FNUAG RZWZH ODAPO RX  
 >>> [K O M]  
 Retour au mode COMMANDES.  
 COMMANDE -> [&]  
 Passage au mode CRYPTAGE.  
 KMB BBB ONJ OUR  
 BONJO URLES AMISQ UIVON TSEFA IREUN EENIG MA  
 >>> [K O M]

Fig.108



"Dormir sur nos deux oreilles." Elle est vraiment stupide cette phrase de Nulentout. Vous avez tenté la chose ? Ou alors c'est que vos deux oreilles elles sont très longues et toutes moles !



## 24) BOUUUMMMMMmmmm ... Collision de PILE.

**P**articulièrément **NON** indispensable, *ce chapitre ne s'adresse qu'aux programmeurs* qui ont été voir dans les démonstrateurs comment sont traitées certaines séquences. Sans que ce soit forcément des exemples à afficher sur les murs, l'intégralité des démonstrateurs a été rédigée avec un souci permanent de rigueur et d'optimisation. S'il est vrai que le problème de collision de **PILE** ne se pose généralement qu'avec des programmes très volumineux qui consomment toutes les ressources internes de l'ATmega328, il peut toutefois survenir avec des logiciels moins développés mais qui utilisent des procédures récursives qui s'invoquent un grand nombre de fois sans sortie.

- **Brusquement un programme présente un comportement anormal,**
  - **La séquence qui diverge n'a rien à voir avec les dernières modifications effectuées.**
- Quand un tel comportement étrange se produit avec la certitude qu'il n'y a aucune relation entre la séquence anormale et les derniers correctifs apportés au logiciel, il faut diagnostiquer une collision de **PILE**, c'est la cause la plus probable du problème.

➤ **La PILE : Tout programmeur doit s'en préoccuper.**

**V**isitons un court instant l'intimité du fonctionnement d'un programme. Chaque fois que le logiciel fait appel à une procédure, le microcontrôleur sauvegarde sur la **PILE** l'adresse de retour pour pouvoir continuer quand cette dernière est terminée. Si une procédure en appelle une deuxième, qui en invoque une troisième, toutes ces adresses de retour s'empilent les unes sur les autres. Puis, en retour de "subroutine" elles sont dépilées et l'emplacement sur la **PILE** libéré. Chaque fois qu'une procédure utilise une variable locale, on réserve de la place sur le **TAS**. Plus il y a de variables locales dans les procédures qui s'invoquent les une et les autres, plus on va entasser des octets de données. Naturellement, en retour de subroutine on libère la place sur le **TAS**. La Fig.109 schématise le fonctionnement interne du microcontrôleur pour la gestion des **variables dynamiques**. Le haut de la zone en vert est réservé à la **PILE** qui progresse "vers le bas". Par exemple en **1** on a empilé deux adresses de retour. Quand la deuxième subroutine se termine, son adresse de retour est récupérée et l'emplacement libéré en **3**. En **2** ont été entassées deux variables locales. En retour d'une subroutine, en **4** la place de la variable qui n'est plus utilisée est libérée. Au cours du programme, la zone non utilisée jaune se rétrécit à chaque appel de procédure ou de fonction, et se réduit pour chaque variable locale. Si l'on a trop de procédures qui appellent des procédures et qui entassent des données locales trop volumineuses, arrive un moment où il y a, comme en **5**, **COLLISION de PILE**. Le **TAS** et la **PILE** se rencontrent, l'un écrasant les données de l'autre. *Le programme devient totalement incohérent et se met à faire n'importe quoi.* C'est un problème particulièrement sournois, car strictement rien ne prévient le programmeur. Par exemple dans une procédure vous remplacez une variable locale qui était un **byte** par un **int**. Imaginons que le contexte à empilé et entassé tellement de données qu'il ne restait plus de place. Maintenant la variable locale fait un octet de plus. Sur un **RUN** se produit alors la collision tant destructrice. Votre programme se met à faire n'importe quoi. Vous avez beau analyser, analyser et analyser encore, le croquis est propre et en aucun cas ne peut expliquer le comportement du logiciel.

**CONCLUSION :** Il faut impérativement s'assurer que ça n'arrivera pas.

➤ **Prendre une assurance vie pour la PILE.**

**P**arer le risque de **COLLISION de PILE** est tellement important, que le C++ d'Arduino met à notre disposition un outil spécial pour évaluer les risques encourus. Aussi, chaque fois que je termine un gros logiciel qui invoque des sous-routines à la pelle, avec des variables

locales nombreuses, avant de considérer qu'il est fiable j'en teste la **PILE**, ou plus exactement l'amplitude de la zone jaune qui reste quand le programme a fait son RESET. Le petit programme qui effectue ce travail consomme environ 140 octets. Aussi, sur un logiciel avec aucun gaspillage il n'y a pas la place. Je me contente de passer en remarque certains affichages, et valide la procédure. Puis elle repasse en remarques et je restitue les affichages. Dans notre application, vu que je gaspille à outrance, avec l'espoir de "rentabiliser" ma carte Arduino NANO, l'outil d'évaluation de la **PILE** est resté actif et de plus encombre le **MENU des OPTIONS**.

### ➤ Configuration qui produit la collision de **PILE** avec le **TAS**.

**C**oncrètement on oublie royalement qu'un nombre important d'interruptions se produisent en tâche de fond. Quelquefois un codeur rotatif génère des interruptions, sans compter les procédures **delay()**, les fonctions telles que **millis()**, la **PWM** ... une foule de ressources internes déclenche des interruptions. C'est transparent car c'est le compilateur C++ qui sur ces instructions fait sa cuisine interne. Arrive un moment, ou trop de données sont empilée sur le **TAS** et viennent écraser les adresses empilées. Puis le délimiteur '}' de fin d'une procédure ou d'une fonction demande au processeur de dépiler une adresse de retour. Comme cette dernière contient les résidus de la variable qui a "écrasé" les octets, le programme se "branche" strictement n'importe où. Étant alors sur du code objet incohérent, le comportement du logiciel devient totalement aléatoire.

**PRÉVENTIF** : Aucun programmeur n'est à l'abri d'une telle "catastrophe". Aussi, pour minimiser les risques il faut placer le minimum de chaînes de caractères dans le programme car en réalité elles sont placées sur le **TAS**. Il faut également (*Et surtout.*) minimiser les tableaux.

**CURATIF** : Quand se produit le **Scratchhh prouitchhh bom bring protchhhh !** c'est qu'il est trop tard. Nous avons placé plein plein plein de bavardages, alors que nous savons que ces "bla bla bla" sont entassés dans la mémoire dynamique. Notre démonstrateur comporte une foule de procédures et de fonctions qui passent des paramètres, sans compter les **for (byte l=1, ...)** qui ne sont pas gratuits. Les variables locales des boucles **for** doivent aussi être logées en RAM. Enfin les tableaux sont gourmands en octets.

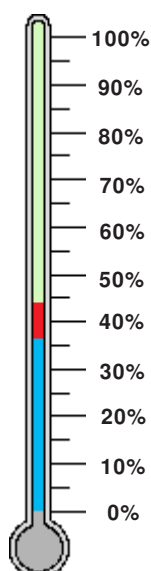
**REMÈDE** : Dégager impérativement de la place sur le **TAS** en minimisant le nombre et la taille des variables locales. Éviter les procédures récursives. Placer les textes en **EEPROM** etc.

### MANIPULATION :

01) Frapper la commande '**p**'. (*C'est tout, du coup ici MANIPULATION est écrit au singulier !*)

Cette option du **Menu de base** affiche la place qui reste entre la **PILE** et le **TAS** au moment où elle est invoquée. Dans notre cas l'espace disponible avoisine 1018 octets. C'est amplement suffisant pour assurer la stabilité de notre logiciel. Une longue expérience dans la pratique des microprocesseurs m'a convaincu qu'à partir d'environ cent octets de disponibles, la collision de **PILE** n'est plus du tout à craindre. Donc avec **P13** on peut rester confiants, il n'y aura pas d'incident de circulation ou ce sera la conséquence d'une vermine dans le programme qui aura échappé aux innombrables vérifications qui ont été effectuées au cours de cette saga.

Fig.110



### ➤ Un dernier petit bilan avec le programme d'exploitation

**C'**est la fin de ce didacticiel, et le programme ultime d'exploitation est arrivé à maturité. Sachant que l'on allait disposer d'autant de place que désiré, plusieurs options "de luxe" ont été ajoutées dans le but d'améliorer la convivialité d'utilisation de ce programme. (*Et également pour rentabiliser l'exploitation des ressources internes de l'ATmega328.*) Dans cette optique, en dernière minute le **Menu de base** a été complété par l'encadré du bas qui précise la fonction des options de la commande '**e**'. À court d'idée pour l'exploitation de notre codeuse virtuelle, il est donc temps de songer à passer à une autre activité. On constate au final, que même avec l'ajout des options de convivialité on n'arrive pas à utiliser la moitié de l'espace réservé au programme dans le microcontrôleur. Aussi, toutes les routines de traitement du signal électrique virtuelles sont écrites et vérifiées et pourront servir éventuellement à créer une réplique matérielle avec des roues qui tournent, un clavier sur lequel faire clic clic clic, un tableau sur lequel brancher des lignes croisées et un tableau de LEDs pour simuler les ampoules électrique. Cette perspective est très alléchante ...

## 25) Changement de programme !

Rédigeant le message secret rose du dernier chapitre, j'ai été obligé de m'y prendre à plusieurs reprises, car j'oubliais un accentué ou une virgule etc. Par ailleurs, je n'ai pas vraiment respecté les protocoles car j'ai doublé d'un 'E' des accentués comme le 'é' par exemple, alors que dans les directives de l'époque seuls 'ä', 'ö' et 'ü' étaient concernés car très fréquents en allemand. Aussi, vu qu'il nous reste de la place à gaspiller, j'ai trouvé utile d'ajouter la commande 'u' dans le **Menu de base** pour nous aider dans ce type de manipulations si on désire vraiment respecter les consignes de l'époque. Le programme **P13** a été remplacé par **P14\_Respect\_des\_PROTOCOLES.ino** dans lequel la COMMANDE 'u' a été ajoutée et remplacera avantageusement son "frère d'arme". La nouvelle fonction se charge de :

- Remplacer la virgule par 'Y' et le point final par 'X'.
- Remplacer le "ch" par 'Q'.
- Remplacer 'ä', 'ö' et 'ü' respectivement par "AE", "OE" et "UE".
- Supprimer les ESPACES.

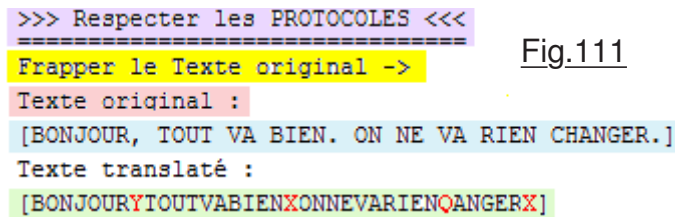
**ATTENTION :** La nouvelle fonction ne gère pas :

- Doubler les noms propres.
- Remplacer les chiffres et les nombres par leur équivalents en toutes lettres.

### MANIPULATIONS :

- 01) Téléverser **P14** dans l'ATmega328 et ouvrir le **Moniteur de l'IDE** : Dans le **Menu de base** vous pouvez observer la nouvelle commande '**U**' ou '**u**'.
- 02) À votre préférence imposer '**b**' si vous désirez le bruiteur.
- 03) Envoyer la commande '**u**' : Le programme attend une chaîne inférieure à 61 caractères
- 04) Commencer par le texte "**Bonjour, tout va bien. On ne va rien changer.**". (*Penser à ce bon vieux **Copier / Coller**.*)

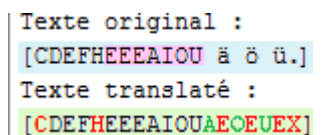
La Fig.111 présente le résultat. Dès que l'on a soumis le '**u**', le titre en violet s'est affiché, complété par le texte d'invite jaune. Puis, la saisie étant terminée, **P14** précise par le texte orange la nature du texte bleu encadré par des crochets. S'il n'y a pas de caractère illégal, le tout est alors suivi de l'équivalent dans la zone verte. Les espaces ont été supprimés et en rouge on peut noter les changements effectués.



```
>>> Respecter les PROTOCOLES <<<
=====
Frapper le Texte original ->
Texte original :
[BONJOUR, TOUT VA BIEN. ON NE VA RIEN CHANGER.]
Texte traduit :
[BONJOURYTOUTVABIENXONNEVARIENQANGERX]
```

Fig.111

05) Maintenant proposer '**u**' suivi de "**cdefhéeèâîôû ä ö ü.**" pour constater l'effet sur les accentués. Dans cet exemple dont la Fig.112 montre le résultat, les accentués "français" dans la zone rose ont simplement été remplacés par leurs équivalents simples. Le 'C' non suivi de 'h' et le 'H' non précédé de 'c' ont été conservés. Le point final est remplacé par 'X' et surtout, les trois lettres avec des trémats ont été affichées sans ce dernier (*Et colorisées en vert.*) et complétées par le 'E'.



```
Texte original :
[CDEFHÉÈÈÂÎÔÛ ä ö ü.]
Texte traduit :
[CDEFHÉÈÈÂÎÔÛAEÖEÜEX]
```

Fig.112

- 06) Poursuivre par '**u**' et le texte "**aaa1bbb**" : Le BIP sonore attire notre attention. Seul le texte qui précède l'erreur est converti. Comme ici l'erreur est le chiffre '**1**', il est converti en '**&**' pour faciliter la saisie des commandes. Donc **P14** ne peut pas savoir si dans la réalité c'est un '**&**' ou un '**1**' qui ont été soumis au logiciel, d'où les deux caractères qui signalent l'intrus.
- 07) Tester avec '**u**' et "**aaa&bbb**" : Même comportement.
- 08) Enfin, proposer une erreur simple comme "**aaa456bbb**" sans oublier au préalable le '**u**'.

Quand un caractère illégal est rencontré, la transposition s'arrête immédiatement, la suite du texte est ignorée. Le caractère fautif est signalé entre crochets. Il ne reste plus qu'à reprendre la suite. Si la ligne était longue, rien n'interdit dans le texte original précisé entre crochet, d'en **Copier** dans l'écran du **Moniteur** la suite non traitée et de la **Coller** ... en ayant éliminé le ou les caractères incorrects.

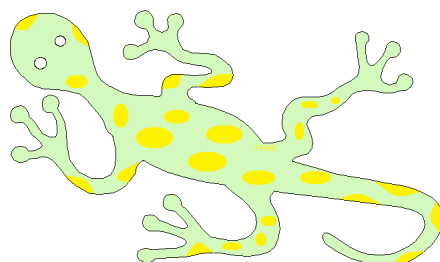
La somme abusive de 736 octets de programme et de 105 octets occupés en mémoire dynamique constituent une exagération pour une fonction qui risque au final d'être relativement peu utilisée. Mais comme je désirais rentabiliser au maximum l'ATmega328, je n'ai pas hésité une seconde à me faire ce petit plaisir. Peut être que cette fois c'est vraiment la fin !



26) Résumé de quelques conseil pour programmer de façon méthodique.

Probablement trop long aura été cette saga informatique. Le but fondamental reste dans la pratique du langage C++ et dans les techniques et les conseils particuliers proposés dans cette narration. Ces points singuliers sont du coup trop éparpillés, et il m'a semblé très utile d'en élaborer ici une liste avant de nous quitter :

- Avant de programmer **il faut absolument consacrer du temps pour déterminer relativement finement ce que l'on désire exactement obtenir** et en prouver la faisabilité.
- La première qualité d'un logiciel quel qu'il soit est sa LISIBILITÉ.
- Les lignes qui doivent facilement se retrouver seront terminées par // @ @ @ @ @ @ @ @ @ @ @ @.
- Placer un maximum de commentaires pour expliciter certains calculs, certains choix etc.
- Pour une lisibilité maximale du listage, **toujours choisir avec beaucoup d'attention les noms des identificateurs des constantes et des variables** avec des noms qui "parlent".
- Toutes les déclarations sont placées en tête de programme.
- Éviter autant que possible les séquences très longues imposant des défilements du listage sur l'écran de l'ordinateur. Une procédure ne devrait jamais dépasser "la taille verticale" de l'affichage.
- Préciser en tête de listage le but précis du programme, ainsi que son comportement attendu.
- Ajouter en tête de listage les branchements à effectuer.
- Préciser en tête de listage la date de dernière modification du programme, la taille du code et celle des données en mémoire dynamise. (*Fonction de la version à préciser du compilateur.*)
- Il est fortement recommandé de placer toute fonction ou procédure avant celle qui y fait appel.
- Pour structurer proprement les constantes et les variables utilisées par le logiciel :
  - \* Les classer par type et globalement par ordre alphabétique.
  - \* Les lister par tailles croissante : **boolean, char, byte, int, long, float, double, tableaux** ...
- **Affectation judicieuse des Entrées / Sorties** : Pas de critères dans ce projet.
- Dans **void setup** initialiser en premier les broches d'E/S. Vers la fin initialiser alors les variables en classant les instructions par l'ordre alphabétique des identificateurs.
- **Optimiser le code doit virer à l'obsession** :
  - \* Optimiser le code c'est **Minimiser la taille du code** objet qui encombrera la mémoire,
  - \* Optimiser le code c'est **Minimiser le temps d'exécution** d'une séquence critique.
- **Optimisation de la taille des variables** : **il est absolument indispensable de choisir le type de chaque variable pour en minimiser la place occupée en mémoire dynamique et en accélérer le traitement.** C'est un impératif absolu. Du bon choix du type des données dépend considérablement la taille occupée par le programme et le temps d'exécution. **ATTENTION**, les tableaux sont les données les plus voraces en espace utilisé et les textes sont intrinsèquement des tableaux. Aussi, **si la mémoire non volatile EEPROM n'est pas utilisée pour mémoriser des données** à conserver sur coupure alimentation, **y logger un maximum de textes** à afficher sur les écrans.
- Pour diverses raisons **la boucle de base doit "tourner" à grande vitesse.** (*Rapidité de prise en compte d'une consigne opérateur, rapidité des affichages etc.*) Le programme étant "terminé", il est fortement conseillé d'en mesurer la fréquence d'exécution.
- Un moyen incontournable pour diminuer les temps de développement consiste à **SIMULER LES DONNÉES** au lieu de les mesurer, et à diminuer certains délais de comportement initialisés.
- Il est conseillé en développement de placer dans la boucle de base un "stéthoscope" pour s'assurer à tout moment que le logiciel n'est pas enlisé dans une séquence très longue ou infinie.
- **Si des paramètres dans des procédures sont susceptibles d'être modifiés** à la mise au point, il est de loin conseillé de **les définir dans des constantes en tête de listage.**



## 27) Les "loupés".

Aucun projet, qu'il soit industriel ou de loisir ne saurait aboutir à la perfection absolue. Entre les désirs initiaux, ce qui était envisagé et ce qui résulte de compromis inévitables, s'insinuera forcément des divergences. (*Sans compter les petites imperfections de programmation.*) Cette modeste Machine Énigma n'échappe pas à ce principe non démontré mais qui frise l'absolu. Par exemple quand on sauvegarde la CONFIGURATION en EEPROM on enregistre un doublon de trois Octets : Les trois premières lettres du GROUPE d'identification qui sont identiques à l'Orientation des trois Rotors. C'est trois Octets de perdus. (*Où à récupérer plus tard si nécessaire.*) Il a fallu composer avec les réalités matérielles, et l'on peut noter à peine un petit "regret" : Il n'a pas été envisagé dès le départ de sauvegarder 13 FICHES croisées en EEPROM. Ensuite cette modification aurait été un peu lourde à effectuer, et j'avoue que j'ai envie de passer à autre chose. Il est du reste assez rare, que sur un projet de cette "envergure" je ne trouve que si peu de points sombres, et aussi dérisoires. Au final, dans cette petite boîte bleue j'ai intégré bien plus de fonctions que je ne l'avais imaginé initialement. À mes yeux, le logiciel correspond largement à ce que je désirais. Franchement je ne vois pas ce que l'on pourrait y ajouter. La seule faiblesse de ce petit bloc bleu, c'est qu'il n'est pas autonome, il lui faut un ordinateur pour dialoguer avec l'utilisateur. Réaliser une version autonome, vla une bonne idée pour reprendre le collier : Créer une réplique matérielle. Ce sera mon prochain projet. Enfin, avant de nous quitter, je ne résiste pas au plaisir de vous soumettre un message ultra secret à décrypter sur notre chiffeuse :

Attention, le contenu de ce document confidentiel respecte les protocoles de l'époque. Pour le décoder il vous faudra au préalable consulter la Fiche n° 14 sur les "*Techniques de cryptage*" disponible dans le fichier FICHES A5.pdf disponible dans le dossier <Documents>.

Pour décoder utiliser la configuration par défaut avec les commandes 'd', suivi de deux fois 'o'. N'oubliez-pas le 't' suivi de '&' et penser à Copier/Coller.

AAA BBB ONJ OUR

QIOTA TKNWV NMNTT LROOP CSLUR GRBGO YRRIT BBLPG DJVKZ NJTTY BSFXS  
KCUHT HOETH QLFBU AAGWB TLYZU KXZQO SYXVE ZXHUO BYNKP NGBGL DCMWL  
HXIPF POVSL KODRH OTJPC DUFJA KFQQP SMLFQ TBHOV PIGDG PGEER QHLYW  
YKTBS UFHJW AXTKQ NMGUW CTEO

Amusez-vous bien, et pourquoi pas se retrouver sur un autre projet ?

*Chères lectrices, chers lecteurs, cette longue présentation arrive à son terme. Tout à une fin, mis à part l'Univers, et arrive forcément un moment où il faut raisonnablement considérer que "le voyage d'agrément" est terminé.*

*Je souhaite intensément que certaines et certains oseront s'engager dans la réalisation d'un clone, je ne doute pas de leur réussite. Surtout, je vous souhaite à toutes et à tous de trouver dans ces lignes le plaisir de la découverte. Si d'aventure vous engagez vos heures de liberté dans une telle réalisation et que vous rencontriez une difficulté, les amis du forum pourront probablement vous aider. Dans le pire des cas, vous pouvez me contacter sur : [michel.droui@laposte.net](mailto:michel.droui@laposte.net) et dans les limites de mon temps de libre, c'est avec grand plaisir que je tenterai de vous dépanner. Je vous souhaite à toutes et à tous agréable lecture et ... que vos secrets militaires soient bien gardés !*

*Chaleureusement : Nulentout.*

