

Machine de TURING Autonome.

Par Nulentout : Mercredi 26 mars 2023.

C chose promise dans le didacticiel de la Machine de Turing Élémentaire, une version entièrement autonome va être développée et va donc faire l'objet de ce deuxième didacticiel. La machine élémentaire réalisée est pratiquement parfaite. D'un coût dérisoire, impliquant un minimum de travail pour la réaliser, minuscule et facile à ranger, elle frise le parfait, et ce d'autant plus qu'elle est très performante et conviviale à utiliser. Elle frise le parfait ... *mais elle n'est pas autonome*. On ne peut pas facilement l'emporter, car un ordinateur doit suivre. Aussi, envisager une version "de luxe" totalement indépendante est très séduisant. Il est clair que coté complexité on change de registre, tant sur le plan informatique que sur le plan matériel. On peut raisonnablement s'engager dans un tel projet si l'on n'est pas trop exigeant pour un afficheur couleur haute définition, un clavier miniature à 40 touches etc. L'entreprise est d'autant plus abordable, que les composants périphériques du commerce sont accompagnés de "bibliothèques" qui en facilitent grandement la programmation.

Cette "version de luxe" est directement descendante de la petite Machine de Turing Élémentaire et en partage une forte proportion d'ADN. La nature des divers MENUS en est directement issue. Pratiquement toutes les procédures de traitement des données binaires du plateau virtuel sont directement récupérées, et ce n'est que l'interface HOMME/MACHINE qui diffère. Aussi, comme une grande majorité de protocole d'utilisation en découle, je vous invite plus que fortement à commencer par tester avec la Machine de Turing Élémentaire qui n'exige strictement aucun investissement mis à part la carte Arduino NANO et vous permettra de surcroît de vous familiariser avec ce type de programmation si particulier et les concepts qui y sont intimement liés.

Contrairement à l'approche adoptée pour le didacticiel sur la Machine de Turing Élémentaire qui a été écrit lorsque le projet était pratiquement à sa version ultime du développement, pour cette deuxième réalisation je préfère adopter la démarche qui était celle du tout premier projet initial qui a conduit à la machine électromécanique citée en introduction du premier tutoriel. *Il est clair que rédigeant les documents au fur et à mesure de l'avancement du projet, il y aura forcément des remises en cause et des changements de stratégie.* Cette façon de vous présenter l'évolution des stratégies, du matériel, des choix visuels donnera l'impression d'un peu de confusion, car les descriptions ne seront pas "linéaires", il y aura des embuches et des chausse-trapes.

Cette approche me semble plus intéressante qu'une description où les choix et les techniques adoptées semblent couler de source. Hors dans la pratique ce n'est pas du tout la réalité. Vous le savez, et chaque fois qu'une personne s'aventure dans un loisir de création, il y a des déconvenues. Comme *ce projet va me servir également à illustrer la programmation avec Arduino*, aussi j'ai décidé de dévoiler non seulement les solutions adoptées, mais également le chemin parcouru pour aboutir. *Il est évident que*

celles et ceux qui ne seront attirés que par l'envie de posséder le petit appareil électronique pourront ignorer somptueusement tous les chapitres de justifications techniques.

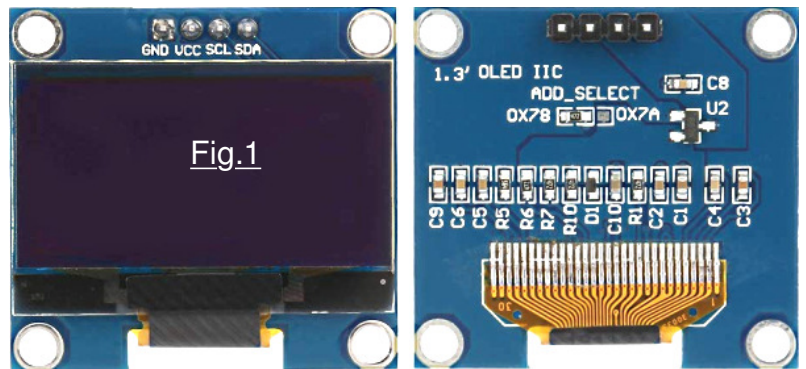


01) Le choix de l'afficheur.

Chaque fois que l'on s'engage dans un projet quel qu'il soit, ce sont les toutes premières décisions qui engagent le plus l'entreprise. On n'a pratiquement rien investi et c'est pourtant ces choix qui auront le plus de conséquences sur la suite par des implications tant "financières" que matérielles et logicielles. Nous allons donc tester chaque périphérique pour valider son aptitude et sa qualité opérationnelle avant de décider définitivement de l'intégrer sur le prototype. C'est la raison pour laquelle nous allons utiliser divers logiciels qui seront considérés comme des outils d'évaluation et des exemples de programmation. Comme périphérique on va utiliser un afficheur numérique. Il est hors de question de se servir de la ligne USB et du **Moniteur de l'IDE** pour dialoguer durant le développement. Il faudrait que le programme intègre la bibliothèque idoine qui consomme déjà environ 1800 octets. Donc, pour pouvoir se faire afficher des données critiques durant les analyses du comportement, l'afficheur est le premier dispositif à brancher sur les sorties du microcontrôleur.

➤ **L'afficheur OLED 1,3 Pouces 128 x 64.**

Bien qu'il présente des faiblesses notables, c'est pour ce périphérique que c'est porté mon choix. Le rafraîchissement est lent, il ne faudra pas espérer voir bouger les pions virtuels en temps réel. Graphique, sa définition est très faible, soit une grille de 128 points en largeur et 64 points en hauteur. Autant dire que sur cette grille on ne pourra pas afficher des textes avec des lignes de 50 caractères. Le nombre de ligne lui même restera dérisoire. Les représentations graphiques seront forcément grossières. Pourtant, montré sur la Fig.1, il surpasse tous ses concurrents potentiels sur un point important : *Son prix d'achat reste assez abordable*. De plus, il est très compact et demeure parfaitement lisible ce qui pour ce projet est appréciable. Il existe en Blanc ou en éclairage bleu. Par exemple vous pouvez l'approvisionner par le commerce en ligne ici :



https://www.amazon.fr/Module-dAffichage-Asixx-Arduino-R%C3%A9solution/dp/B0707D5VJ3R/?ref=sr_1_116&keywords=ded+arduino&qid=166866418&qu=eyJxcm2MOi1Lj4iwcXNhjpNC43NishFzcChjQUlQIC%3D%3D&sr=8-116

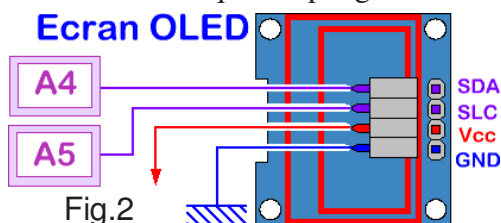
(Croquis : Terme qui désigne un programme en C++ d'Arduino.)

Il reste à en évaluer la pertinence, c'est l'objet de ce chapitre et des croquis associés.

➤ **Mise en œuvre de l'afficheur.**

Outre son prix d'achat abordable, *il ne monopolise que deux broches d'interfaçage sur le microcontrôleur*. Les branchements se réduisent au plus simple puisque l'on réunit directement broche à broche. Comme je sais par expérience que diverses E/S seront impérativement réservées potentiellement à d'autres périphériques spécifiques, ce sont les broches analogiques **A4** et **A5** qui vont être initialisées en fonctionnement BINAIRE. En respectant le schéma de la Fig.2 il ne restera plus qu'à déclarer la bibliothèque **U8glib** pour pouvoir le programmer aisément. Pour vous éviter des recherches fastidieuses sur Internet, cette bibliothèque est disponible dans le dossier qui accompagne ce didacticiel <Machine AUTONOME\Les bibliothèques\U8glib>.

Déclarer une bibliothèque n'est pas très compliqué. Il suffit de cliquer sur l'onglet [Croquis] puis sur [Importer bibliothèque ...], [Ajouter bibliothèque ...] et *indiquer le chemin du dossier* précisé ci-avant. Cette bibliothèque met à notre disposition un grand nombre de procédures. Il est évident que le programme en cours de développement va en consommer à profusion. Il est



donc incontournable d'en connaître l'utilisation. Aussi, comme en permanence on aura besoin de s'y référer, je vous propose dans le même dossier **Bibliothèque U8glib.pdf** qui est un petit livret dont la technique pour le concrétiser est indiquée dans le chapitre 11 de la page 39.

➤ Se familiariser un peu avec l'afficheur.

Avant de foncer tête baissée dans le logiciel qui animera notre dispositif électronique, il me semble judicieux d'ouvrir le livret de la bibliothèque **U8glib** et d'en expérimenter certaines méthodes. À travers quelques petits croquis de démonstration on va se faire une petite idée des possibilités offertes par cet afficheur. Ensuite, il faudra élaborer "le scénario" global du projet, c'est à dire définir les grandes lignes de la façon dont on va installer l'interface HOMME / MACHINE. Dans le dossier <Les programmes Arduino> on commence par téléverser le croquis **P01_Test_de_textes.ino** sur la carte Arduino dont la ligne série est réunie à un port USB de l'ordinateur. Les quatre lignes électriques sont réunies conformément à la Fig.2 à la carte NANO qui pour le développement est enfichée sur une plaquette d'essais. Ce premier exemple sert à vérifier la

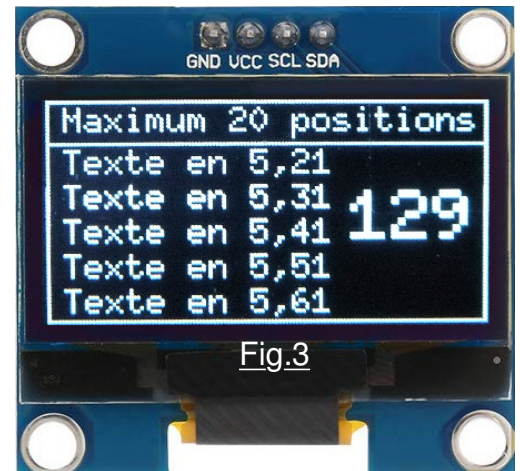


Fig.3



Fig.4

lisibilité des textes les plus petits et évaluer les possibilités d'affichage.

Le croquis comporte peu d'instructions. Ces dernières utilisent des méthodes incontournables, à analyser avec les explications du petit manuel. Pour continuer, téléverser l'exemple **P02_Graphismes_et_textes.ino** qui intègre à la fois des méthodes graphiques et des affichages de textes. Il permet d'estimer la rapidité ou la lenteur pour tracer des entités géométriques. Bien que le programme ne comporte de quelques lignes de code, on constate que 30% de l'espace de programme est déjà consommé.

C'est la faille lorsque l'on utilise un afficheur graphique. Il s'octroie la part du lion dans l'espace de programme. On se doute qu'il va falloir optimiser à outrance le code pour gérer l'application. Avec **P03_PAPILLON.ino** on se fait un petit délire, et surtout on peut analyser la façon dont un dessin se construit "par tranches" comme précisé en page 13 du livret sur la bibliothèque **U8glib**. Enfin, avec **P04_Parametrer_Interlignage_par_calcul.ino** on termine cette familiarisation partielle avec les méthodes de la bibliothèque de l'afficheur.

C'est la faille lorsque

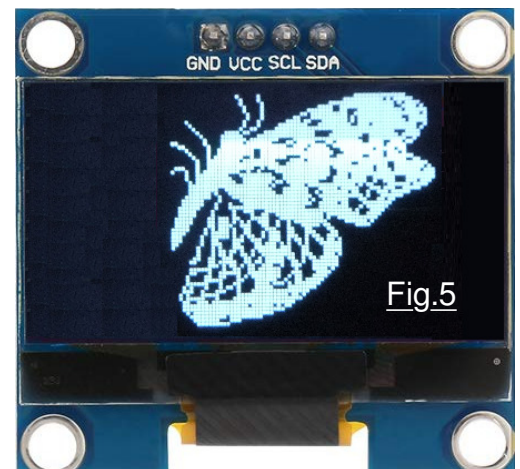


Fig.5

02) Encodeur rotatif KY-040.

Supérieur en agrément d'utilisation, un bouton rotatif est bien plus adapté pour la gestion de menus qu'un clavier avec divers touches consacrées "aux déplacements" dans les textes des items. Dans ce type d'application confinée dans un boîtier de petites dimensions, j'intègre régulièrement ce type de capteurs. Facile à se procurer sur la toile, il suffit de proposer "encodeur rotatif KY-040" à un quelconque moteur de recherche pour obtenir une foison de références. Le KY-40 est un codeur **sans butée** à 20 points par tour pourvu d'un "bouton poussoir" par appui sur la tige de commande centrale. La sortie se fait par deux lignes pilotées par des capteurs de type codage Gray. (*Algorithmes Utilisateur n°7 par exemple.*)

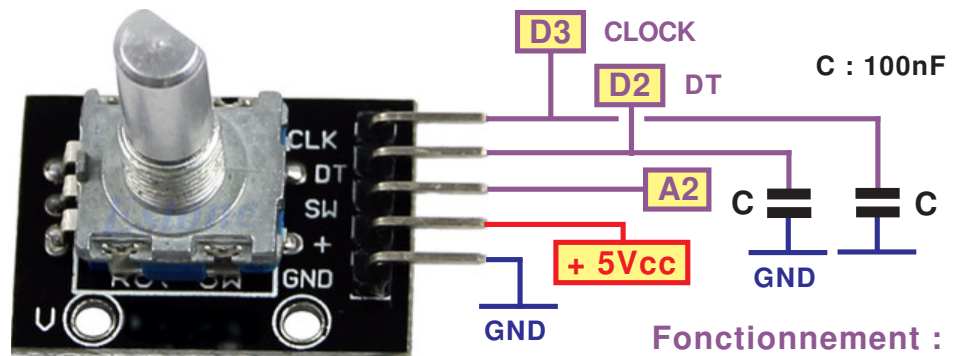
Caractéristiques techniques du module KY-040 :

- Consommation maximale : 10 mA sous 5 Vcc.
- Température de fonctionnement : - 30 à + 70 °C.
- Température de stockage : - 40 à + 85 °C.
- Durée de vie du capteur de rotation : Minimum 30 000 cycles.
- Durée de vie du contact central : Minimum 20 000 cycles.
- Résistance de passage du contact de RAZ : 3 Ω maximum.

La Fig.6 donne le schéma des branchements à réaliser sur Arduino NANO pour gérer les menus dans notre projet.

L'entrée **A2** est choisie, car elle ne peut fonctionner qu'en entrée.

Fig.6



Concrètement les trois sorties sont alimentées au **+ 5 Vcc** par des résistances de **10 kΩ**. La sortie **SW** est celle de l'inverseur piloté par appui sur la tige du rotor. Les deux sorties **CLK** et **DT** sont en réalité deux sorties classiques avec déphasage de 90° souvent nommées **A** et **B** pour ce type de capteur. Le déphasage de 90° électriques des signaux **CLK** et **DT** permet de déterminer le sens de rotation. (Voir Fig.7) **Le capteur est traité par interruptions.**

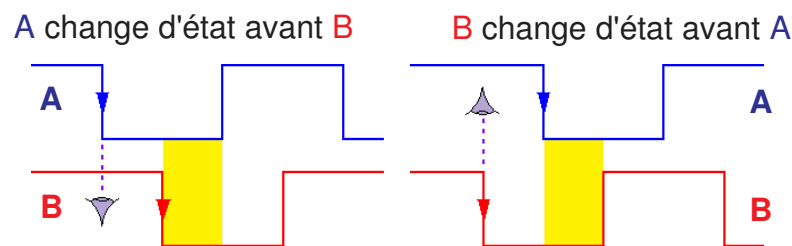


Fig.7

➤ **Mise en œuvre de l'encodeur KY-040.**

Disponibles à des tarifs très variables dans le commerce en ligne, la qualité de ces composants est également fonction de l'investissement financier. Aussi, j'ai approvisionné des exemplaires dont le coût est très faible, mais de très médiocre qualité pour les contacts électriques du codeur. Il en résulte des "faux contacts" qui engendrent des "indexages" complètement aléatoires. Un moyen radical de parer ce problème et peu coûteux consiste à ajouter un condensateur **C** de 100nF entre **A**, **B** et **GND**. Si le composant que vous avez approvisionné donne pleine satisfaction, souder ces deux condensateurs sur le circuit ne sera pas utile. Par contre, si sur les deux programmes de validation **P05_Indexer_dans_un_MENU.ino** et **P05_Indexer_un_Emplacement.ino** le déplacement de l'index est illogique, alors les deux condensateurs **C** de la Fig.6 seront indispensables.

Avec **P05_Indexer_dans_un_MENU.ino** on teste la faculté d'indexer un **MENU de BASE**. Ce petit logiciel ajoute à la gestion de l'afficheur OLED celle du capteur rotatif utilisé en interruptions. Ce programme met en place le **MENU de BASE** et les aiguillages pour l'indexation des items. Si on clique sur le **Bouton Poussoir Central** il y a validation de l'item indexé. Dans la suite **Bouton Poussoir Central** sera remplacé par les initiales **B.P.C.** Tant que le **B.P.C.** est appuyé, la LED d'arduino s'illumine. Le sous-menu activé ouvre un écran noir avec sa nature. Cliquer à nouveau sur le **B.P.C.** ramène au **MENU de BASE**. Le Menu **BARILLET** est en place. Valider l'Item engendre un BIP sonore et ramène au **MENU de BASE**.

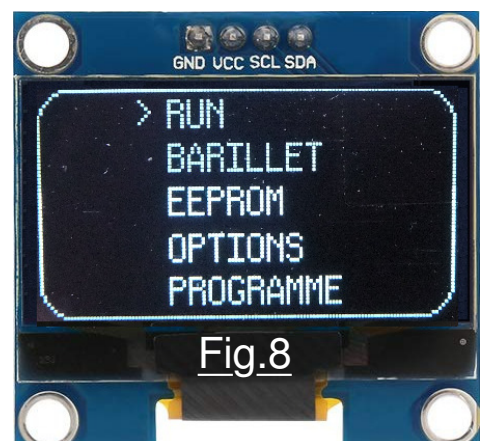


Fig.8

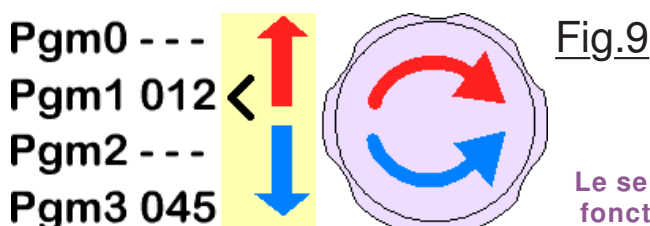
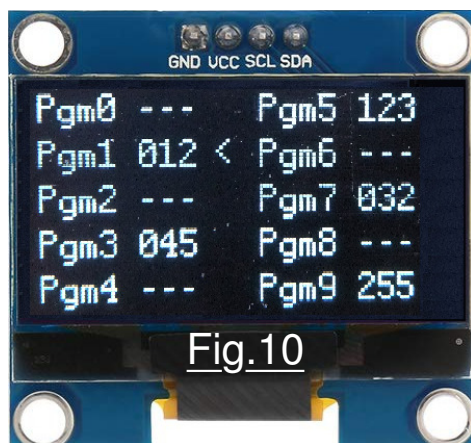


Fig.9

NOTE : Seul le **MENU de BASE** montré sur la Fig.8 sera encadré pour le mettre en évidence.

Le sens de déplacement vertical du curseur en fonction du sens de rotation du codeur rotatif est montré sur la Fig.9 ci-contre.

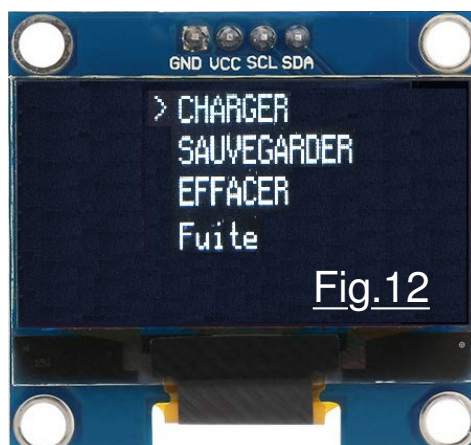
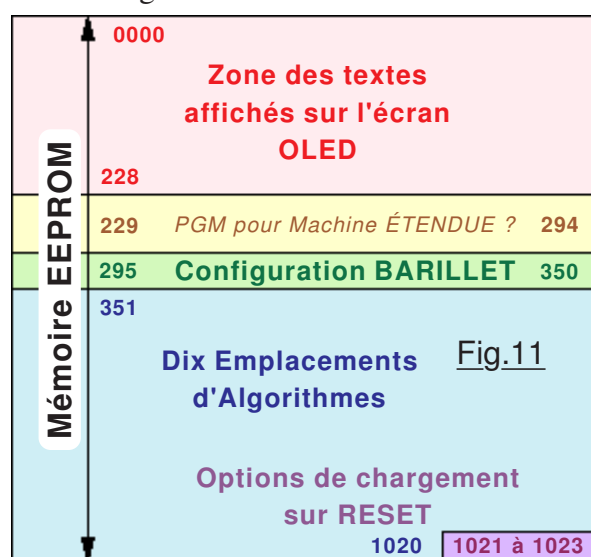


T éléverser [P06_Indexer_dans_un_Emplacement.ino](#) va préparer l'avenir en tenant compte de la faible définition de l'afficheur. Ce croquis visualise comme montré sur la Fig.10 le contenu fictif de l'EEPROM qui ne contiendra que dix algorithmes au maximum. Cette limitation est adoptée pour deux raisons. La première vient du fait qu'afficher plus d'emplacements simultanément sur l'écran imposerait l'utilisation d'une police de caractères bien plus petite, la lisibilité deviendrait médiocre. *(Et votre narrateur a dépassé les 74 printemps !)*

La deuxième raison est issue de l'expérience acquise au cours de mes nombreux projets. L'utilisation d'un afficheur graphique impose des méthodes extrêmement boulimiques en octets de programme. Pour un projet comme celui-ci, il est déjà évident qu'il va falloir optimiser à outrance.

➤ Stratégie d'utilisation de l'EEPROM.

D irectement influencée par les choix effectués pour l'élaboration de la machine élémentaire, les 10 algorithmes envisagés à ce stade des études seront logés en "haut" de la mémoire non volatile EEPROM. Juste "en dessous" on réservera la place pour une configuration de BARILLET. On se réserve la possibilité de charger un algorithme sur RESET et une configuration de BARILLET, qui comme pour la machine élémentaire, s'octroient les trois derniers octets. On ne s'interdit pas la possibilité de sauvegarder un algorithme étendu, comme montré sur la Fig.11, mais ce ne sera possible que si les procédures indispensables ne saturant pas l'espace disponible pour le programme. Comme l'afficheur consomme un espace programme sur l'ATmega328 considérable, il va falloir optimiser à outrance, contrairement à ce qui avait été fait pour la machine élémentaire. Aussi, l'un des moyens le plus efficace pour "tasser" le code consiste à loger les textes affichés en EEPROM. Nous allons donc réserver le début de l'EEPROM dans ce but. La suite du développement orientera les solutions adoptées et influencera, ça on le sait, les performances au sens général de la petite machine électronique. Le programme [P06_Indexer_dans_un_Emplacement.ino](#) intègre le codeur rotatif et ajoute l'indexation des dix fichiers potentiels. Si on Clique sur le B.P.C. l'index pointant un fichier vide représenté par "---" il y a génération d'un BIP d'alerte. Cliquer sur une position occupée indiquée par la référence numérique de l'algorithme ne provoque aucune action. Tant que le B.P.C. est appuyé, la LED d'arduino sur **D13** s'illumine.



➤ La gestion des divers menus.

Q uel que soit le système étudié, la convivialité d'utilisation et la qualité opérationnelle seront directement influencées par la facilité de "se déplacer" dans les divers menus. Aussi, avant de mettre en place les fonctions qui effectuent les traitements idoines, [Arduino\P07_Noyau_de_base.ino](#) architecture le programme squelette et émule les menus principaux. Ce démonstrateur intègre l'affichage de P05 et celui du contenu de l'EEPROM testé dans P06. Dans l'affichage du contenu de l'EEPROM on indexe un fichier. Quand on clique sur le B.P.C, si cette position est vide il y a retour au **MENU de BASE** avec génération d'un BIP d'avertissement. Si

l'emplacement est occupé il y a ouverture du **Menu EEPROM**. (Voir la Fig.12) Le BP central déclenche la fonction qui dans ce croquis n'effectue aucun traitement. Toutefois elle

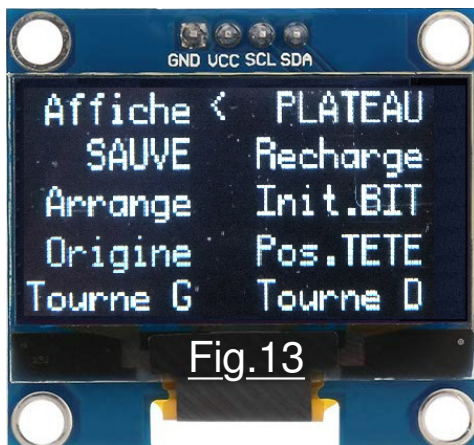


Fig.13

est présente, et il suffit de "remplir" les corps des procédure au fur et à mesure que sera développé le programme. Le menu de la Fig.13 est relatif à la gestion du BARILLET. Chaque clic sur l'un des dix items génère un BIP sonore suivi d'un retour au **MENU de BASE**. La Fig.14 montre le **Menu PROGRAMME**. Comportement analogue à celui du menu précédent, sauf que la Fuite ne génère pas le BIP sonore. L'ossature du logiciel d'exploitation est en place. La compilation précise que le code binaire consomme déjà 10348 octets soit 33% de la place totale disponible pour loger le programme. C'est assez inquiétant, car on ne fait que choisir des actions, et encore pas toutes. (*Les OPTIONS ne sont pas encore agencées car elles seront fonction de la place restant disponible pour le code.*) Les procédures de traitement sont vides. Les traitements de l'algorithme ne sont pas implémentés. Bref, l'avenir est problématique, et il ne sera peut être pas possible de faire tout ce qui est implémenté sur la **Machine Élémentaire**. Il est tout à fait possible qu'il soit nécessaire de renoncer à du graphique par exemple. Bref, le but de tous ces outils consiste à valider les approches, les choix, et surtout d'optimiser le "source" au fur et à mesure. C'est d'autant plus facile, que l'on ne se préoccupe que de sections bien déterminées du logiciel.

➤ Passage des textes en EEPROM.

Première étape avant de pouvoir utiliser les textes, il faut les inscrire dans la mémoire non volatile du microcontrôleur. Dans ce but, on téléverse **P00_Initialiser_EEPROM.ino** qui n'avait pas été utilisé. Comme suggéré dans le programme à la ligne 70 on valide la remarque pour remplir l'EEPROM de "FF" faciles à repérer sur le listage. Puis on active le programme qui affiche les chaînes de textes actuellement inscrits. La Fig.15 liste ceux qui sont présents à ce stade du développement avec leurs adresses. Ce croquis installe également une configuration BARILLET et dix algorithmes pour pouvoir

Programmes en EEPROM :		
=====		
Emplacement 01	->	Programme 001
Emplacement 02	->	Programme 002
Emplacement 03	->	Programme 003
Emplacement 04	->	Programme 001
Emplacement 05	->	Config.BARILLET.
Emplacement 01	->	Programme 046
Emplacement 02	->	Programme 055
Emplacement 03	->	Programme 028
Emplacement 04	->	Programme 034
Emplacement 05	->	Programme 054
Emplacement 06	->	Programme 029
Emplacement 07	->	Programme 052
Emplacement 08	->	Programme 018
Emplacement 09	->	Programme 012
Emplacement 10	->	Programme 053

Fig.16

continuer le développement et disposer de données à traiter. La Fig.16 représenterait ce que montrerait dans ces conditions le programme de la **Machine Élémentaire**. En vert, les cinq emplacements qui n'existent plus. Concrètement, le n°5 devient une zone de 56 octets placée juste avant les algorithmes. *Les deux références 254 et 255 n'étaient pas utilisables pour les programmes, maintenant elles ne sont plus utiles et peuvent être "récupérées".* La Fig.17 présente l'implantation actuelle en EEPROM. La zone jaune sera affectée à un algorithme pour Machine ÉTENDUE s'il est possible d'ajouter cette option. La zone blanche correspond à de la place pour loger encore du texte. En vert se prouve l'emplacement réservé pour loger une configuration de BARILLET. Enfin toute la zone bleue est occupée par les dix

algorithmes potentiels avec en violet tout en haut le la mémoire les trois octets de gestion des rechargements durant le RESET. La mémoire EEPROM étant gavée de toutes ces données, il faut maintenant téléverser **P08_Textes_en_EEPROM.ino** le démonstrateur strictement identique à P07 dans lequel, seuls les affichages qui étaient directement dans le programme sont

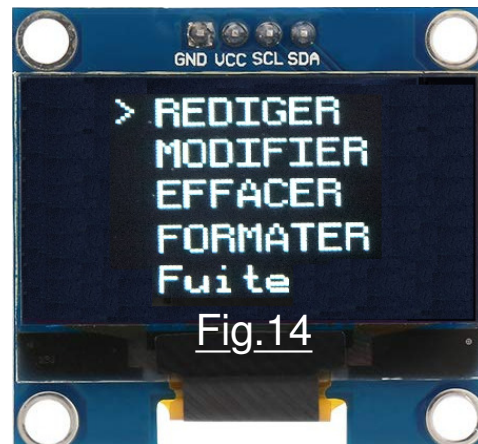


Fig.14

PTR = 0	Affiche	PLATEAU
PTR = 18	SAUVE	RECHARGE
PTR = 34	Arrange	Init.BIT
PTR = 52	Origine	Pos.TETE
PTR = 70	Tourne G	Tourne D
PTR = 89	CHARGER	
PTR = 96	SAUVEGARDER	
PTR = 107	EFFACER	
PTR = 114	Fuite	
PTR = 119	OPTIONS	
PTR = 126	REDIGER	
PTR = 133	MODIFIER	
PTR = 141	FORMATER	
PTR = 149	RUN	
PTR = 152	BARILLET	
PTR = 160	EEPROM	
PTR = 166	PROGRAMME	
PTR = 175	---PgmPGM	
PTR = 185	Transferer ?	

Fig.15

ADRS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ADRS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
0000	41	66	66	69	63	68	65	20	20	20	20	50	4C	41	54	45	0512	14	04	14	04	11	07	11	07	11	07	11	08	11	08	11	08	
0016	41	55	53	41	55	56	45	20	20	20	52	45	43	48	41	52	0528	12	09	12	09	12	09	12	0A	12	0A	12	0A	14	0B	14	0B	
0032	47	45	41	72	72	61	6E	67	65	20	20	20	49	6E	69	74	0544	14	0B	14	08	14	08	14	08	22	09	07	11	02	10	02	10	
0048	2E	42	49	54	4F	72	69	67	69	6E	65	20	20	20	50	6F	0560	03	00	00	10	00	09	04	09	04	08	04	08	05	00	00	00	
0064	73	2E	54	45	54	45	54	6F	75	72	6E	65	20	47	20	20	0576	00	08	06	00	00	0C	00	01	0B	01	0B	00	0B	08	08	00	
0080	20	54	6F	75	72	6E	65	20	44	43	48	41	52	47	45	52	0592	00	08	00	11	09	11	09	10	09	10	0A	00	00	00	00	10	
0096	53	41	55	56	45	47	41	52	44	45	52	45	46	46	41	43	0608	06	00	00	14	00	00	00	00	00	00	0C	36	00	02	00	00	
0112	45	52	46	75	69	74	65	4F	50	54	49	4F	4E	53	52	45	0624	08	00	0A	03	00	0C	00	00	08	00	10	04	10	04	12	05	
0128	44	49	47	45	52	4D	4F	44	49	46	49	45	52	46	4F	52	0640	00	0C	00	00	10	00	08	02	00	00	00	00	00	00	00	00	
0144	4D	41	54	45	52	52	55	4E	42	41	52	49	4C	4C	45	54	0656	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0160	45	45	50	52	4F	4D	50	52	4F	47	52	41	4D	4D	45	20	0672	00	00	00	00	00	00	00	00	00	00	00	00	00	00	1D	00	
0176	2D	2D	2D	50	67	6D	50	47	4D	54	72	61	6E	73	66	65	0688	02	00	02	08	00	09	07	09	03	00	0B	08	00	08	00	08	
0192	72	65	72	20	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	0704	04	08	00	08	00	12	05	10	00	10	00	10	06	10	00	10	
0208	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	0720	00	0A	02	08	00	08	00	08	08	08	00	08	00	14	09	10	
0224	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	0736	00	10	00	10	0A	10	00	10	00	0C	02	00	00	00	00	00	
0240	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	0752	0C	34	12	02	10	00	08	06	10	00	00	00	10	03	10	00	
0256	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	0768	00	00	0C	04	08	00	00	00	08	05	08	00	10	01	10	01	
0272	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	0784	08	00	14	07	10	0A	10	00	00	00	10	08	10	00	00	00	
0288	FF	FF	FF	FF	FF	FF	FF	02	02	01	01	01	01	01	01	01	0800	0C	09	08	00	00	00	08	06	10	00	00	00	10	01	00	00	
0304	01	01	01	01	01	01	01	01	01	01	01	01	01	01	02	01	01	0816	00	00	00	00	12	10	02	11	00	00	00	08	04	08	04	00
0320	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	0832	07	10	04	10	00	08	06	12	05	14	03	0C	06	10	00	10	
0336	01	01	01	01	01	01	01	02	02	02	02	02	02	02	02	2E	0848	04	12	04	08	00	08	00	10	01	00	00	00	00	00	0C	00	
0352	01	03	01	05	02	07	02	09	04	0A	04	0B	02	0B	02	0B	0864	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0368	01	0A	01	09	02	08	01	07	04	05	04	03	02	02	02	01	0880	00	00	00	00	00	00	00	00	0C	00	02	00	00	08	00	08	00
0384	01	01	01	01	02	01	02	02	02	04	03	04	04	02	05	02	07	0896	00	00	10	03	11	04	00	00	00	08	10	00	00	00	10	05
0400	01	09	01	0A	02	0B	02	0C	04	0C	04	0C	02	0B	02	0A	0912	12	00	0C	06	0C	06	08	00	08	00	08	07	08	00	00	00	
0416	01	08	37	00	02	00	00	08	00	08	00	00	00	08	03	12	0928	10	03	10	09	10	09	10	00	10	00	10	00	10	0A	00	00	
0432	00	10	00	10	04	0A	05	10	00	00	00	00	00	08	00	08	0944	00	00	00	0C	00	00	00	00	00	00	35	12	02	10	00	10	
0448	06	12	07	08	00	10	09	00	00	10	00	10	08	0A	05	10	0960	00	08	03	00	00	08	05	0A	04	08	00	08	00	10	01	00	
0464	00	0A	05	00	00	14	00	00	0C	00	00	00	00	00	00	00	0976	00	10	08	00	00	0C	00	08	06	09	07	09	00	08	00	10	
0480	00	00	00	00	00	1C	11	06	11	02	10	00	11	03	11	03	0992	01	00	00	00	0B	00	00	14	00	10	09	00	00	00	0A	10	
0496	11	03	11	04	11	04	11	04	12	05	12	05	12	05	14	04	1008	00	00	01	11	00	00	00	00	0C	00	00	10	00	FF	09	FF	

Fig.17

maintenant puisés en EEPROM. On compile le code source, on le téléverse sur la carte ARDUINO, et l'afficheur OLED avec le codeur rotatif doivent présenter un comportement strictement identique à celui de P07. On note que la taille du logiciel passe de 10348 octets à 10422 soit une taille plus importante de 74 octets. Tout ce travail semble donc totalement négatif. C'est une fausse impression, car le code a été "plombé" des méthodes de `EEPROM.h` qui de toute façon sera indispensable pour échanger des données avec la mémoire non volatile. Il convient maintenant de continuer à définir les stratégies à privilégier et de compléter la structure matérielle et la structure logicielle.

03) Configuration matérielle et scénario envisagés.

À ce stade du développement, il faut définir la structure du clavier qui servira à l'interface HOMME/MACHINE. Plus il sera "discret", et meilleure sera la mise en œuvre de la machine. Une bonne expérience sur ce type de réalisations de faible encombrement m'a convaincu qu'associer au codeur rotatif un clavier à cinq touches est généralement suffisant. Sans préjuger des proportions et des dimensions, la Fig.18 donne une idée de ce que pourrait être l'appareil quand il sera terminé. Étant droitier, je privilégie la position du capteur rotatif à droite de l'afficheur OLED. Une diode

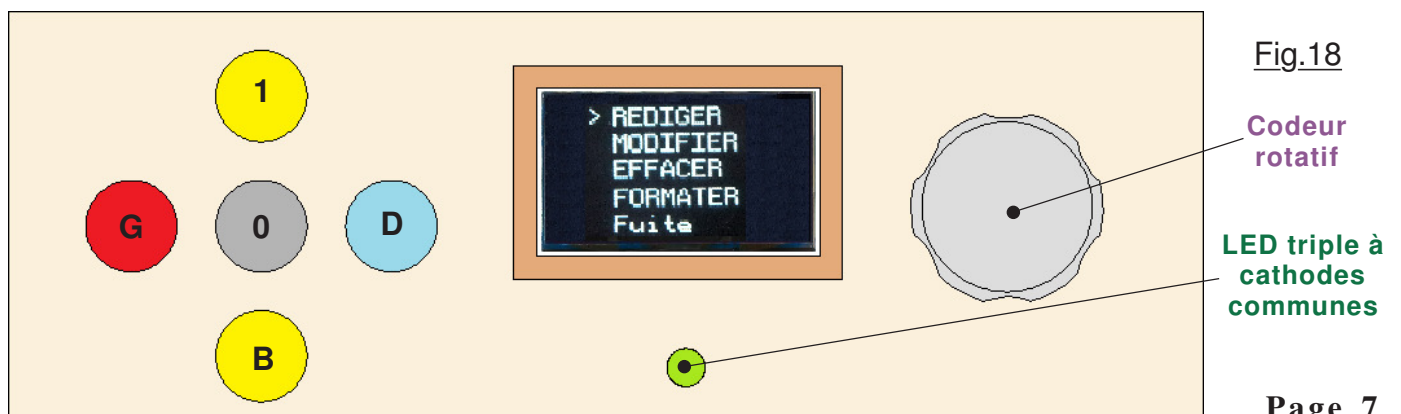


Fig.18

Codeur rotatif

LED triple à cathodes communes

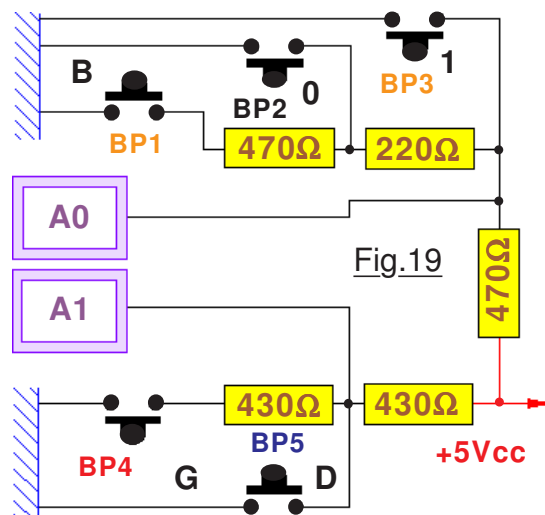
électroluminescente avec trois couleurs fondamentales sera largement suffisante pour informer l'opérateur de certains événements particuliers. Quand au clavier, il reste à définir ses protocoles d'utilisation. Par exemple **G** et **D** peuvent choisir des sens de rotation, **B**, **0** et **1** imposer une écriture. Ou les quatre BP extérieurs servent à se déplacer dans une grille, et le codeur rotatif à changer la valeur de la cellule pointée. Ces façons d'utiliser le clavier élémentaire seront choisies par expérimentation pour aboutir à des protocoles conviviaux et une machine à forte qualité opérationnelle.

> L'agencement matériel.

Il n'est pas inutile de redécouvrir le monde. Le schéma électrique du clavier montré en Fig.19 reprend intégralement un circuit totalement analogue utilisé sur d'autres projets qui sont du reste en ligne sur ROBOT MAKER. Les cinq boutons poussoir seront de taille à déterminer, le choix des éléments étant probablement influencé par leur disponibilité dans les tiroirs. L'approvisionnement sera d'autant plus aisé que je m'impose des types à simple fermeture de contact. Ce sont de loin les plus courants et existent à profusion sur les étagères virtuelles du commerce en ligne.

Le clavier à cinq touches.

Compte tenu du faible nombre de touches, il est plus rentable de monopoliser deux entrées analogiques que d'effectuer un multiplexage. La Fig.19 présente la solution retenue dans laquelle les valeurs des résistances sont choisies pour minimiser le courant consommé, optimiser les zones de décision et offrir une immunité satisfaisante aux parasites. Les valeurs des résistances sont choisies de façon à avoir des écarts de tension "homogènes" en fonction des



A0	Repos	BP1	BP2	BP3
Tension	5v	3,1v	1,9v	0
CAN	1023	608	323	0

Fig.20

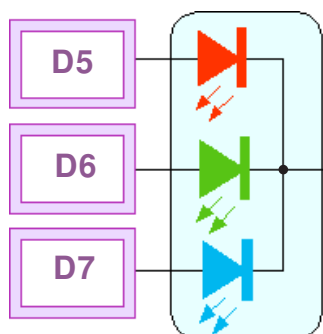
A1	Repos	BP4	BP5
Tension	5v	2,6v	0v
CAN	1023	511	0

Fig.21

numérisation effectuée par les convertisseur de l'ATmega328. On y trouve les seuils de comparaison pour déterminer quel est le bouton poussoir utilisé. Pour optimiser le filtrage des parasites, les seuils de comparaison pour déterminer le bouton poussoir actionné sont placés à exactement la moyenne entre les valeurs typiques issues des CAN. (CAN : Conversion Analogique Numérique.)

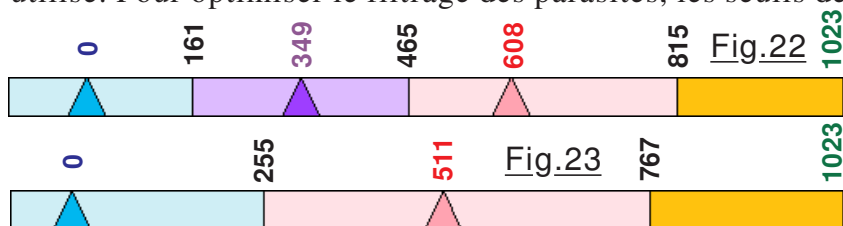
La diode électroluminescente.

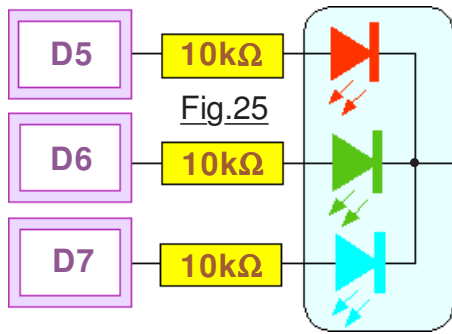
Fig.24



Composant ultra courant, la LED triple à cathode commune est ultra courante et très populaire. Il existe diverses références dont les rendements peuvent varier. Celle approvisionnée donne de très bons résultats avec une résistance de limitation commune de 10kΩ. Non seulement elles présentent une très bonne luminosité avec un courant très faible, et surtout, la clarté est analogue sur les trois couleurs fondamentales. On peut alors n'introduire qu'une seule résistance de limitation de courant **R1** entre cathodes et **GND**.

Cet agencement ne convient que si une seule couleur est utilisée à la fois. Dans le cas contraire, c'est la couleur qui engendre la chute de tension aux bornes la plus faible qui s'allumera à





l'exclusion des autres. Si on désire allumer deux ou trois LEDs simultanément, il faut les piloter conformément à la Fig.25 chaque ligne étant indépendante. Par exemple rouge et bleu donne du violet, bleu et vert produit du cyan, rouge et vert ressemble à du jaune. Les trois simultanément produisent une sorte de blanc moiré. On peut aussi produire des effets particuliers, par exemple on allume bleu et on fait clignoter rouge etc. Comme pour notre application nous n'avons pas encore déterminé quel sera l'usage final, lors du développement effectué par des branchements provisoires sur des plaques à essais on adopte le schéma de la Fig.25 quitte à simplifier comme sur la Fig.24 si les LEDs ne sont utilisées qu'indépendamment les unes des autres.

➤ L'agencement logiciel.

L'étape qui suit naturellement pour la détermination du matériel consiste à mettre en place le code pour gérer les nouveaux composants réunis au microcontrôleur. Et aussi prendre en compte les données logées en EEPROM, en particulier ne plus simuler comme sur la Fig.10 et effectuer une lecture réelle du haut de la mémoire non volatile pour en déduire le contenu. C'est précisément l'objet de **P09_Noyau_Complet.ino** que l'on téléverse sur la carte NANO pour en observer le comportement et surtout vérifier les routines de servitude. Typique d'un assemblage électrique provisoire, la Fig.26 présente ce qui sera probablement la version définitive du matériel. Les fils électriques qui relient les composants sont tellement nombreux, que la **Carte Arduino NANO** insérée sur une plaque à essais est pratiquement invisible masquée sous le fatras des connexions électriques volantes. On comprend que la réalisation d'un petit circuit imprimé adapté sera plus que salutaire. Du reste, pour pouvoir effectuer sans trop se contorsionner, le mini **bouton de RESET** de la carte Arduino a été doublé par un élément extérieur bien dégagé de ce méli-mélo inextricable. Ce tutoriel s'adresse à toutes celles et tous ceux qui désirent concrétiser une petite Machine de Turing autonome sans pour autant avoir du temps à consacrer au plaisir la programmation. "Oubliez" dans ce cas *ce qui suit*, car destiné aux internautes qui utilisent régulièrement l'**IDE** pour leurs propres projets :

Quand nous engageons régulièrement nos loisirs dans le développement de petites applications à microcontrôleurs, on finit souvent par employer fréquemment des périphériques courants. La concrétisation du projet peut prendre plusieurs semaines. Aussi, il ne faut pas hésiter à s'entourer de petits accessoires qui serviront régulièrement et nous assisteront tout au long de nos expériences de programmeurs. Par exemple la **Colonne** place bien au dessus du montage l'**Afficheur** qui est sur un support articulé pour l'orienter en hauteur directement vers l'opérateur. En particulier on remarquera que la **Ligne OLED** qui relie l'**Afficheur OLED** au microcontrôleur est longue et souple. On peut ainsi bien dégager l'ensemble, voir l'approcher de l'opérateur. Ainsi, à l'aide

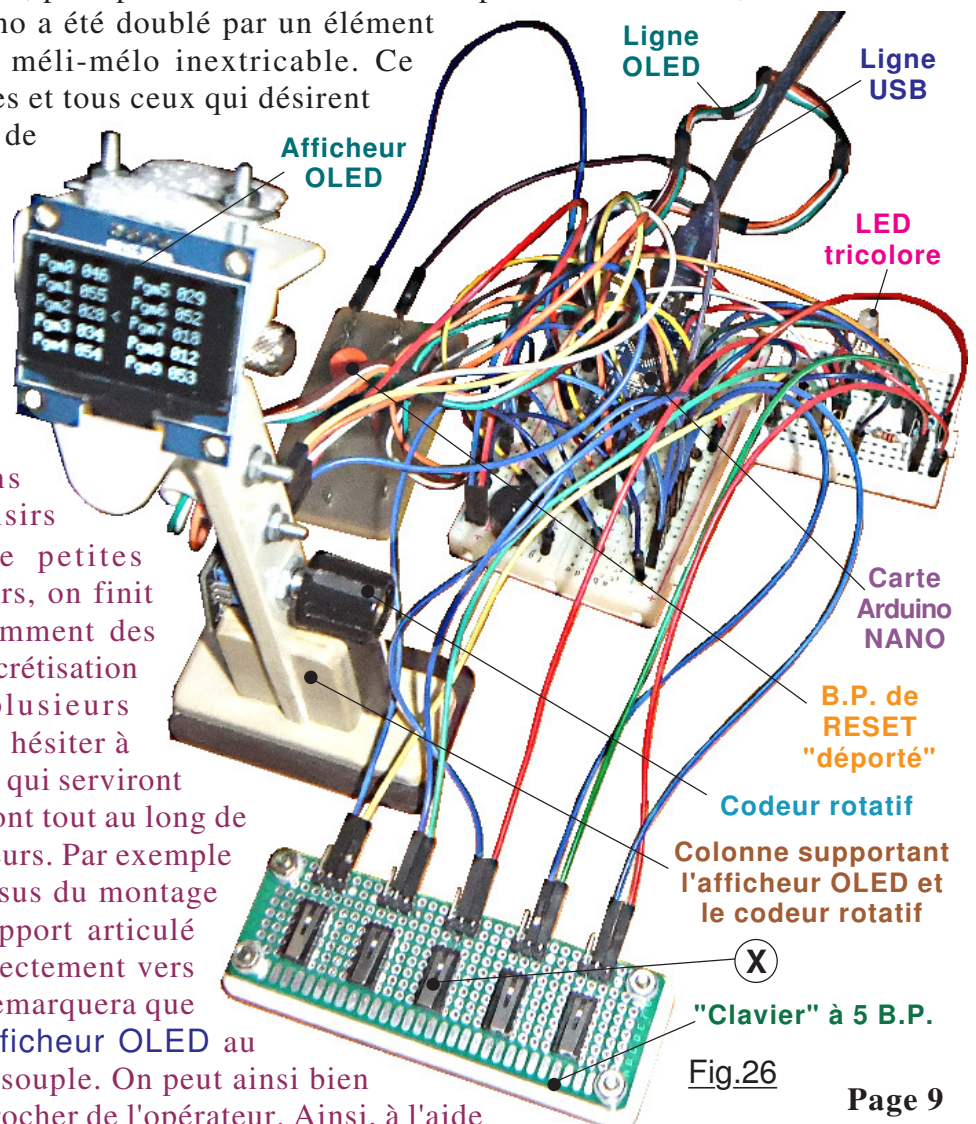


Fig.26

d'une loupe à fort grossissement on peut bien voir certaines zones de l'écran. Dans cette application ce ne sera certainement pas le cas. Mais j'utilise également des afficheurs bien plus petits de 0,96 pouces de diagonale. Pour vérifier des dessins, des graduations etc, il faut zoomer. Sur cette **Colonne** est également immobilisé le **Codeur rotatif** sur lequel on clique un nombre incalculable de fois. Il importe que ce type de composant soit vraiment aisé à manipuler. Il en va également de même pour les mini-claviers qui équipent nos petites réalisations. Clipser des petits boutons poussoir sur des plaques à essais est une solution très malcommode. Les fils de liaison gênent l'accès, et quand on clique pas exactement au milieu ils basculent et il faut alors les réinsérer correctement sur la plaquette expérimentale. C'est ainsi que dans le tiroir des outils on trouve un clavier à 16 touches pour multiplexer, des inverseurs et boutons en tout genre, des rampes de LEDs et toute une famille de petites électroniques d'accastillage. Bref, au cours de vos "cheminements" de programmeur, construisez vos outils ...

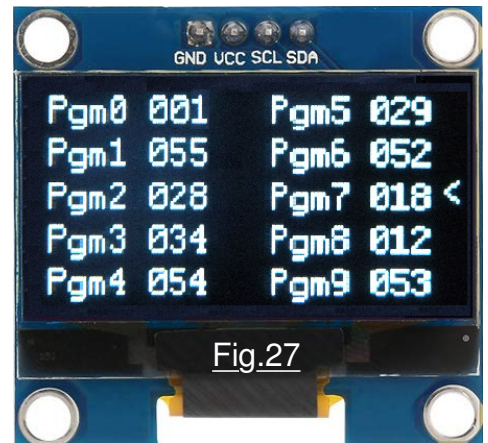


Fig.27

Fin des "confidences", revenons à **P09_Noyau_Complet.ino** qui sur un RESET affiche le **MENU de BASE**. À l'aide du codeur rotatif indexer l'item **EEPROM** et valider. S'affiche l'écran de la Fig.26 sur lequel on a indexé **Pm7** qui contient l'algorithme dont la référence est le n°18. Il ne s'agit plus d'une simulation, car avec P09 on effectue réellement l'analyse du contenu de l'EEPROM. (Notez au passage que les dix emplacements sont occupés pour pouvoir librement effectuer des actions en tous genres dans la mémoire non volatile et en vérifier le bon déroulement. C'est du reste le bon moment pour ouvrir une parenthèse relative aux croquis fournis.)

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.

Cliquer deux fois sur le B.P.C. afin de revenir au **MENU de BASE** et vous en profitez pour constater que maintenant c'est la composante rouge de la LED qui s'allume durant l'enfoncement. Puis cliquer sur l'un des "Switch" qui ici sert de bouton poussoir. Par exemple sur **X** de la Fig.27 et immédiatement, comme le montre la Fig.28 la valeur de la numérisation est affichée dans l'encadré rouge. C'est cette valeur, ainsi que celles des quatre autres boutons qui seront prises en compte pour optimiser les seuils de détection qui sur les figures 22 et 23 sont relatifs à des résultats issus d'un clavier installé sur un autre projet. Les résistances sont de valeurs précises, alors que sur le montage de la Fig.26 c'est du "n'importe quoi". Le logiciel détermine immédiatement quel est le bouton poussoir actionné. Vous avez certainement remarqué que tant que le switch est activé, la composante verte de la LED triple qui s'allume. Il y aura donc un contrôle visuel de la prise en compte de ces deux périphériques. Pour achever cette expérimentation, on clique sur tous les boutons et surtout on retient :

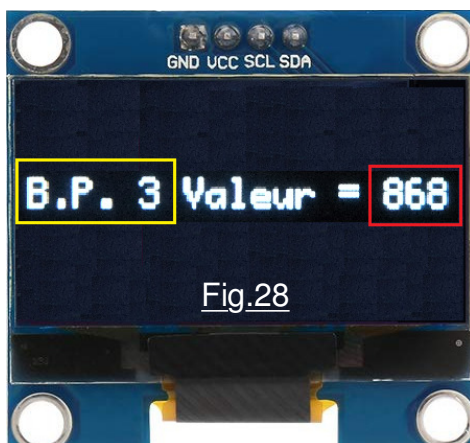


Fig.28

Quand le clavier définitif sera réalisé avec ses résistances sélectionnées avec soin, on réutilisera P09 pour mesurer les valeurs réelles sur le prototype et on calculera les valeurs optimisées pour les seuils de détermination des B.P.

BILAN : Le compilateur s'active sur **P09** et annonce son verdict : Le programme actuel fait 11270 octets et occupe déjà 36% de la place disponible pour le logiciel alors que l'on effectue encore aucun traitement. Pour le moment il ne faut pas s'affoler. En effet, une longue expérience a montré que c'est toujours la mise en place des procédures de base qui gloutonne un maximum de place. Ensuite l'évolution sera moins dramatique. Et puis s'il faut réduire nos objectifs on taillera dans les prévisions quitte à abandonner les belles représentations graphiques. De toute façon, à partir du moment où l'on désire utiliser un composant de faible dimensions pouvant afficher un nombre significatif de caractères textuels, qu'il soit graphique ou non n'intervient pas vraiment dans la taille occupées par les procédures puisées dans sa bibliothèque. En revanche, plus on va chercher à bénéficier des possibilités de **U8glib**, plus la boulimie en octets va s'amplifier. Aussi, à ce stade du développement on doit impérativement effectuer des choix et définir une stratégie :

➤ **Stratégie d'utilisation de la librairie U8glib.**

Livret Bibliothèque **U8glib** en main, on notera en bas de **P4** que chaque police de caractère encombrera la RAM dynamique. Il ne faut donc pas s'étonner qu'actuellement le programme du petit livret en consomme 13%. Ce n'est pas dramatique. D'un autre côté, si l'on change plusieurs fois de police de caractères, ce sera chaque fois du code qui s'ajoute. Aussi, comme l'on affichera que des textes "ordinaires", **on n'utilisera qu'une seule police de caractères pour tout le programme**. Par ailleurs, il aurait été tentant d'exploiter les beaux curseurs de la page **P9** du petit livret. Toutefois, il suffit de déclarer la "fonte réduite" et de faire afficher une fois le curseur pour que le programme enfle de 100 octets. **On se passera de la possibilité d'afficher des curseurs graphiques**. Par ailleurs, si possible **on minimisera les outils graphiques utilisés**. Le plan budgétaire étant approuvé, on peut passer au développement du logiciel. Comme l'exploitation de l'appareil sera impossible sans la facilité de créer et de modifier des Algorithmes, la prochaine étape va donc consister à mettre en place l'ÉDITEUR de PROGRAMME ainsi que ses fonctions d'affichage. C'est parti ...

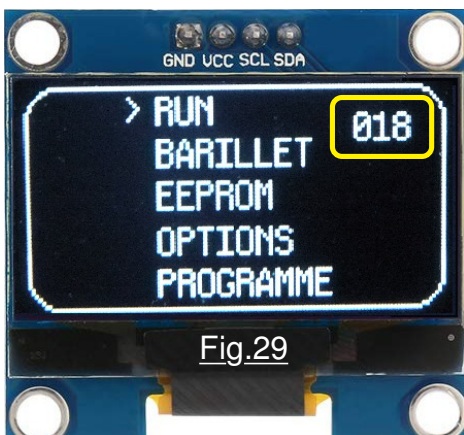
04) L'ÉDITEUR de PROGRAMME.

Clef de voute de l'interface HOMME/MACHINE, il conditionne intégralement l'agrément d'utilisation de la machine pour tout ce qui concerne l'élaboration et la mise au point d'un algorithme. Que ce soit par l'entremise du moniteur de l'ordinateur dans la version élémentaire, ou du petit écran OLED en version autonome, dans les deux cas il doit :

- Permettre de rédiger l'algorithme, (*Avec un éditeur de texte spécifique.*)
- Accepter de modifier n'importe quelle ligne individuellement,
- Pouvoir intercaler à convenance du code dans une ligne vide,
- De lister l'ensemble pour pouvoir le relire facilement et l'analyser en vue de déverminer.

➤ **Lister le programme actuellement en mémoire.**

Développer les fonctions de l'éditeur sera facilité par la faculté de lister ce qui est présent en mémoire dédiée. Aussi, la première fonction à assurer consiste à pouvoir transférer l'un quelconque des algorithmes présent en EEPROM. Puis on va élaborer un scénario de visualisation de la table des transitions. Lorsque l'on pourra afficher facilement chaque ligne de la grille virtuelle de programme, alors il sera pertinent d'élaborer la rédaction et les modifications. Comme toutes ses actions sont connexes, l'affichage des lignes de programme devra si possible être analogue à celui des modifications. Le démonstrateur élaboré pour valider cette facette des études logicielles se nomme **P10_Afficher_un_ALGORITHME.ino** que l'on téléverse pour en expérimenter le comportement. Ce démonstrateur est déjà très avancé, car il intègre des fonctions complexes d'affichage. Par ailleurs, il a été très travaillé pour trouver des méthodes d'utilisation simples et d'un usage "naturel". Il n'est pas évident de faire émerger des protocoles "instinctifs" avec un clavier aussi rudimentaire. Nous allons donc commencer par



les écrans d'affichage de l'algorithme présent dans la mémoire dédiée. On peut déjà observer que le **MENU de BASE** de la Fig.8 est maintenant complété dans l'encadré jaune de la Fig.29 par l'indication en haut et à droite de la référence du programme en cour d'édition. Dans cet exemple on a chargé l'emplacement **Pgm7** de la Fig.27 proposée en page 10. On peut en déduire que l'item **CHARGER** de la Fig.12 est implémenté. Si cette référence est **000** c'est que le programme en mémoire centrale est effacé. Le transfert depuis l'EEPROM tient compte de la possibilité de passer en **Machine ÉTENDUE**. Provisoirement, en cliquant sur l'item **OPTIONS** on inverse le type d'algorithme. Si le logiciel gère cette possibilité de **Machine ÉTENDUE** la référence



Fig.30

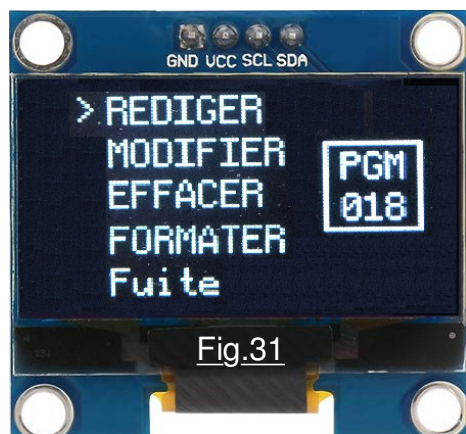


Fig.31

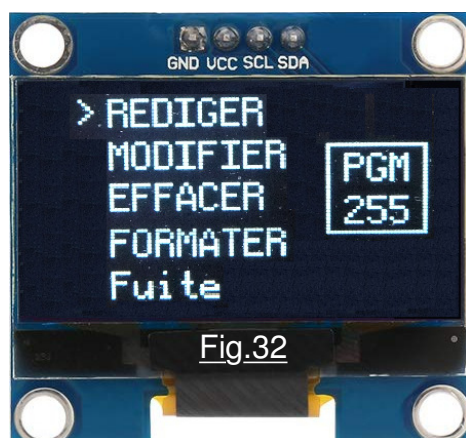


Fig.32

affichée sur la Fig.30 est alors **255**. C'est une référence réservée. Passez à des Algorithmes de 20 transitions possibles n'efface pas celui qui serait en mémoire. Ainsi on peut récupérer un programme standard et le compléter à convenance. On constate sur la Fig.31 que le menu **PROGRAMME** s'est enrichi d'un petit cadre intérieur qui visualise la référence du programme affectée par l'opérateur et sur la Fig.32 celle affectée automatiquement par le logiciel quand on est en mode **Machine ÉTENDUE**.

Lorsque l'on valide la fonction **REDIGER**, on passe directement en saisie des instructions si le mode **Machine ÉTENDUE** est actif. Dans le cas contraire, ce que montre la Fig.33 l'opérateur doit choisir une référence quelconque comprise entre **1** et **254**. La saisie sera limitée à 254 et il sera impossible de proposer 255 la référence réservée. *Si on propose 0 comme référence il y a fuite* avec génération de trois BIPs sonores d'alerte. La fonction **MODIFIER** n'efface pas le programme et ne demande pas sa référence. On débute par la saisie de l'ordre de la transition à partir

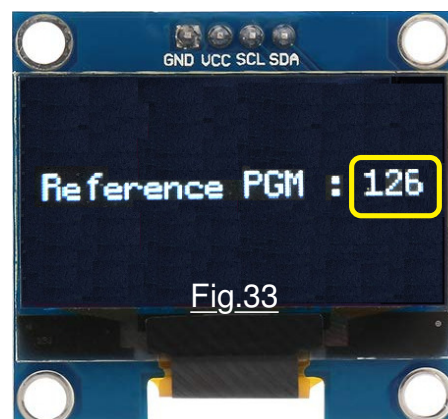


Fig.33

de laquelle on va effectuer les corrections. Si l'opérateur se trompe et propose **0** il y aura correction à partir de la transition n°1. Comme il y aura au maximum 20 transitions possibles et que c'est le bouton rotatif qui sert à indiquer la valeur, il n'y a pas de coefficients "multiplicateurs" comme dans le cas de la saisie de la référence de l'algorithme. (*Un tour de codeur incrémente 20 fois.*)

➤ Protocole de saisie de la référence de l'algorithme.

Cliquer sur le B.P.C. en standard validera la valeur saisie. Toutefois enfoncer la touche centrale du mini-clavier aura le même effet. Comme déjà précisé, proposer **0** engendre la Fuite sans modifier l'existant. Tourner le codeur rotatif dans un sens ou dans l'autre incrémentera ou décrémentera la valeur affichée. Si l'on désire la valeur 238, à raison de 20 incréments par tour la manipulation va s'avérer bien indigeste. C'est la raison pour laquelle le petit clavier va introduire à convenance le choix des coefficients multiplicateurs de la Fig.34 avec, il faut le savoir, des effets de bord. Par exemple avec le coefficient 100, partant de 1, en 3 indexations on arrive à ~~301~~ > **45**. Comme le logiciel n'introduit pas de butée numérique, et que la variable est un **int**, 301 - 256 donne les 45 affichés. Ce n'est pas du tout gênant. La pratique montre qu'avec les

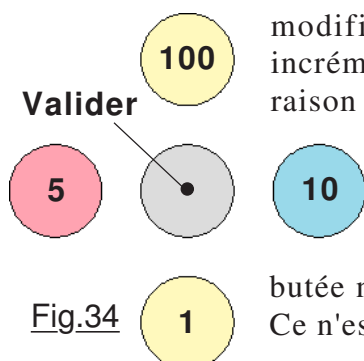
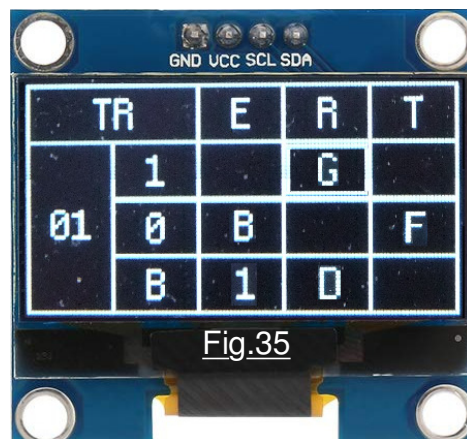


Fig.34

coefficients adoptés en quelques clics on obtient rapidement n'importe quelle référence. Pour la saisie des valeurs pour les instructions de l'algorithme, un protocole spécifique est également élaboré et directement influencé par l'affichage de l'état actuel de la grille virtuelle de programme. Commençons par expliciter la façon dont sera présentée la feuille perforée virtuelle.

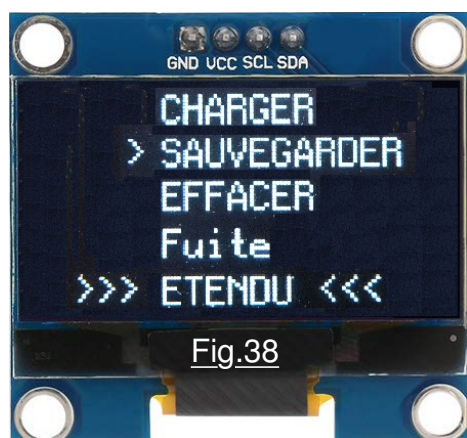
➤ L'affichage du programme de l'algorithme.

Optimiser l'utilisation de la surface de l'afficheur OLED installé sur l'appareil et tenir compte de sa faible définition a conduit après plusieurs expérimentations à privilégier la solution montrée sur la Fig.35 qui sur une page-écran présente les trois lignes d'instructions d'une transition complète. On remarque que l'une des cellules est "surlignée" par le doublement de son encadrement. C'est cette cellule qui sera modifiée quand on tourne le codeur rotatif. Il n'a pas été prévu de fonction ~~LISTAGE~~ car elle n'apporterait strictement rien de plus que la fonction REDIGER ou la fonction MODIFIER. En effet, ces deux dernières permettent facilement de se "déplacer" dans les transitions vers le début ou vers la fin avec les touches "jaunes" du petit clavier. La touche centrale octroie dans ce but la faculté de *ne pas afficher le surlignage* de la cellule pointée ou de *le rétablir*. Du coup, pour explorer l'ensemble de l'algorithme on invoque REDIGER ou MODIFIER, on clique sur le bouton central du



clavier pour masquer l'index, et l'on explore les transitions avec les touches "jaunes". C'est vraiment très convivial, assez intuitif, et cette technique a donc été retenue. La Fig.36 présente l'usage du petit clavier et la Fig.37 symbolise le déplacement du curseur quand les touches rouges et bleues sont cliquées. La bleue engendre un saut du curseur vers la droite. Quand il est sur la colonne T il descend d'une ligne et va à gauche.

Quand sur la ligne du B il est à droite, il revient en haut et à gauche. La touche rouge génère des mouvements symétriques. Avec ces deux touches on peut donc rapidement balayer les neuf cellules de la transition en cours de traitement. On se doute que le menu EEPROM fait appel aux mêmes fonctions pour le mode standard ou le mode Machine ÉTENDUE. Toutefois, dans le deuxième



cas l'affichage du menu est complété comme on peut le constater sur la Fig.38 par l'affichage de son statut particulier. Ainsi l'opérateur est averti de l'état du logiciel. Il n'écrasera pas par erreur un algorithme à 20 transitions par un programme standard. La fonction EFFACER ne sera pas non plus validée par erreur sur l'algorithme le plus long à éditer. Il convient à ce stade de l'étude du croquis P10_Afficher_un_ALGORITHME.ino d'aborder le protocole de modification des instructions.

➤ Protocole de saisie des instructions.

Chaque cellule sera modifiée individuellement. Avec les touches rouge et bleue on indexe la case à modifier. En tournant le codeur rotatif dans un sens ou dans l'autre, on fait changer le contenu de la cellule dans l'ordre ou en rétrograde des possibilités concernant sa nature. Une cellule vide signifie pas de "perforation virtuelle". On ne peut insérer que des instructions logiques et les contradictions ne sont pas possibles. Par contre une écriture redondante ne sera pas signalée à la saisie. Cliquer sur le B.P.C. en standard fait sortir de l'ÉDITEUR de PROGRAMME.

► Optimisation du contenu de l'EEPROM.

Dans la mesure où l'option de Machine ÉTENDUE est validée, car tous comptes faits elle n'engendre pas une consommation démente d'octets de programme, on n'empiètera pas dans la zone **encadrée** de la Fig.39 qui est réservée aux algorithmes de 20 transitions. Optimiser l'utilisation de la mémoire non volatile consiste à en employer la plus grande partie. Pour ce projet il ne sera pas possible de faire mieux, car 100% des 1024 octets servent à préserver des données. Le contenu listé

ADRS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ADRS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
0000	50	4C	41	54	45	41	55	20	20	20	20	20	20	4D	6F	74	0512	14	04	14	04	11	07	11	07	11	07	11	08	11	08	11	08	
0016	69	66	53	41	55	56	45	20	20	20	52	45	43	48	41	52	0528	12	09	12	09	12	09	12	0A	12	0A	12	0A	14	0B	14	0B	
0032	47	45	41	72	72	61	6E	67	65	20	20	20	52	75	6E	20	0544	14	0B	14	08	14	08	14	08	22	09	07	11	02	10	02	10	
0048	3E	50	4C	54	4F	72	69	67	69	6E	65	20	20	20	50	6F	0560	03	00	00	10	00	09	04	09	04	08	04	08	05	00	00	00	
0064	73	2E	54	CA	54	45	52	41	5A	4F	55	49	4E	4F	4E	54	0576	00	08	06	00	00	0C	00	01	0B	01	0B	00	0B	08	08	00	
0080	6D	70	20	53	61	74	75	72	65	43	48	41	52	47	45	52	0592	00	08	00	11	09	11	09	10	09	10	0A	00	00	00	00	10	
0096	53	41	55	56	45	47	41	52	44	45	52	45	46	46	41	43	0608	06	00	00	14	00	00	00	00	00	00	15	36	00	02	00	00	
0112	45	52	46	75	69	74	65	4F	50	54	49	4F	4E	53	52	C9	0624	08	00	0A	03	00	15	00	00	08	00	10	04	10	04	12	05	
0128	44	49	47	45	52	50	47	4D	4F	44	49	46	49	45	52	46	0640	00	15	00	00	10	00	08	02	00	00	00	00	00	00	00	00	
0144	4F	52	4D	41	54	45	52	52	55	4E	42	41	52	49	4C	4C	0656	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0160	45	54	45	45	50	52	4F	4D	50	52	4F	47	52	41	4D	4D	0672	00	00	00	00	00	00	00	00	00	00	00	00	00	00	1D	00	
0176	45	20	2D	2D	2D	50	67	6D	54	72	61	6E	73	66	E9	72	0688	02	00	02	08	00	09	07	09	03	00	0B	08	00	08	00	08	
0192	65	72	54	72	61	6E	73	69	74	69	6F	6E	20	3A	20	52	0704	04	08	00	08	00	12	05	10	00	10	00	10	06	10	00	10	
0208	E9	66	E9	72	65	6E	63	65	20	6C	65	20	C9	54	45	4E	0720	00	0A	02	08	00	08	00	08	08	08	00	08	00	14	09	10	
0224	44	55	20	21	03	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	0736	00	10	00	10	0A	10	00	10	00	0C	02	00	00	00	00	00	
0240	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	0752	15	34	12	02	10	00	08	06	10	00	00	00	10	03	10	00	
0256	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	0768	00	00	0C	04	08	00	00	00	08	05	08	00	10	01	10	01	
0272	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	0784	08	00	14	07	10	0A	10	00	00	00	10	08	10	00	00	00	
0288	FF	FF	FF	FF	FF	FF	FF	02	01	01	01	01	01	01	02	01	0800	0C	09	08	00	00	00	08	06	10	00	00	00	10	01	00	00	
0304	01	02	02	02	02	01	01	01	01	02	01	01	02	01	01	01	0816	00	00	00	00	12	10	02	11	00	00	00	08	04	08	04	00	
0320	01	02	01	01	02	01	01	01	01	02	01	01	02	01	01	01	0832	07	10	04	10	00	08	06	12	05	14	03	0C	06	10	00	10	
0336	01	02	02	02	02	01	01	02	01	01	01	01	01	01	02	01	0848	04	12	04	08	00	08	00	10	01	00	00	00	00	00	15	00	
0352	12	02	12	02	12	02	14	03	14	03	14	03	12	04	12	04	0864	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0368	12	04	14	05	14	05	14	05	12	06	12	06	12	06	14	07	0880	00	00	00	00	00	00	00	00	00	00	00	00	00	08	00	08	00
0384	14	07	14	07	12	08	12	08	12	08	14	09	14	09	14	09	0896	00	00	10	03	11	04	00	00	00	08	10	00	00	00	10	05	
0400	12	0A	12	0A	12	0A	14	01	14	01	14	01	00	00	00	00	0912	12	00	0C	06	0C	06	08	00	08	00	08	07	08	00	00	00	
0416	00	00	37	00	02	00	00	08	00	08	00	00	00	08	03	12	0928	10	03	10	09	10	09	10	00	10	00	10	00	10	0A	00	00	
0432	00	10	00	10	04	0A	05	10	00	00	00	00	00	08	00	08	0944	00	00	00	15	00	00	00	00	00	00	35	12	02	10	00	10	
0448	06	12	07	08	00	10	09	00	00	10	00	10	08	0A	05	10	0960	00	08	03	00	00	08	05	0A	04	08	00	08	00	10	01	00	
0464	00	0A	05	00	00	14	00	00	15	00	00	00	00	00	00	00	0976	00	10	08	00	00	0C	00	08	06	09	07	09	00	08	00	10	
0480	00	00	00	00	00	1C	11	06	11	02	10	00	11	03	11	03	0992	01	00	00	00	0B	00	00	14	00	10	09	00	00	00	0A	10	
0496	11	03	11	04	11	04	11	04	12	05	12	05	12	05	14	04	1008	00	00	01	11	00	00	00	00	15	00	00	10	00	02	09	FF	

Fig.39

PTR = 0 PLATEAU Motif
 PTR = 18 SAUVE RECHARGE
 PTR = 34 Arrange Run XALT
 PTR = 52 Origine Pos. FEE
 PTR = 70 RAZ
 PTR = 73 QUINONTmp Saturer
 PTR = 89 CHARGER
 PTR = 96 SAUVEGARDER
 PTR = 107 EFFACER
 PTR = 114 Fuite
 PTR = 119 OPTIONS
 PTR = 126 RÉDIGER
 PTR = 133 **PG**
 PTR = 135 MODIFIER
 PTR = 143 FORMATER
 PTR = 151 RUN
 PTR = 154 BARILLET
 PTR = 162 EEPROM
 PTR = 168 PROGRAMME
 PTR = 177 ---Pgm
 PTR = 184 Transférer
 PTR = 194 Transition :
 PTR = 207 Référence
 PTR = 217 le
 PTR = 220 ÉTENDU

Fig.40

en Fig.39 est définitif, car les textes de la zone marron sont figés et il ne sera plus possible d'en ajouter. Pour gagner un octet, (*Économies bout de chandelle !*) Le **M** de **MODIFIER** sert à la fin du texte **PGM**. Du coup, ce sigle qui était en adresse 182 (*Voir la Fig.15*) passe maintenant en 133. Au final, dans le programme il a fallu corriger toutes les adresses des textes qui suivent !

Comme les textes figés en EEPROM sont codés en ASCII, il est possible sans aucune pénalité sur la taille occupée, d'introduire à notre guise des accentués. Dans ces conditions ce serait dommage de ne pas en profiter. Aussi, on peut observer dans les textes de la Fig.40 que l'intégralité des caractères qui pouvaient être accentués l'ont été, y compris sur les lettres majuscules. En violet ont été ajoutés les symboles de trois espaces qui sont dans les textes mais non visibles au listage. C'est du reste grâce à l'économie "bout de chandelle" que le mot **ÉTENDU** rentre exactement dans la place qui restait disponible. À partir d'ici, **les nouveaux textes consommeront de la place programme, il ne faudra surtout pas en abuser**. Les "bavardages" seront dorénavant réduits strictement à leur juste nécessaire ...

05) Les fonctions EEPROM.

Pouvoir sauvegarder le programme qui vient d'être saisi en mémoire non volatile semble un complément logique à intégrer dans le démonstrateur **P10_Afficher_un_ALGORITHME.ino** ainsi que la faculté de pouvoir libérer un emplacement en EEPROM. Du coup, toutes les fonctions du menu **EEPROM** seront émulées. Par ailleurs, ce démonstrateur intègre la fonction **EFFACER** du menu **PROGRAMME**. Aussi, pour terminer ce dernier, il est pertinent d'ajouter également la fonction **FORMATER** qui effectue un travail très important en langage Turing au bénéfice du programmeur.

➤ Stratégie d'utilisation du codeur rotatif et du clavier.

Chaque fois que le B.P.C. est appuyé, la couleur rouge de la LED triple s'allume. Tant que l'une des touches du MINI-clavier est cliquée c'est la composante verte qui éclaire. *Si appuyer sur le B.P.C. ou l'une des touches du clavier ne provoque pas d'éclairement, c'est que le périphérique n'est pas pris en compte à ce stade de l'utilisation de la machine.* Généralement, quand on doit valider l'Item d'un menu, c'est avec le bouton central du codeur rotatif. Par contre, *lorsque le programme est en attente d'une touche clavier* pour passer à la suite, ou pour confirmer une intention comme sur la Fig.41 *la composante verte de la LED tricolore clignote rapidement.* L'opérateur sait alors sans

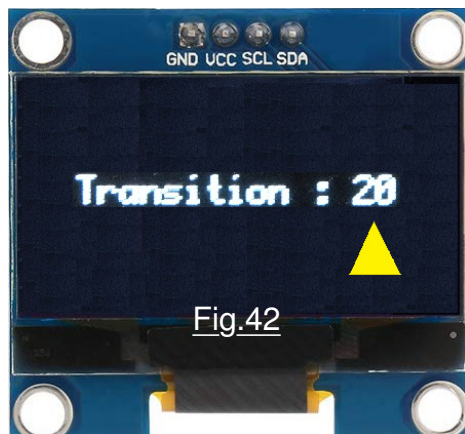


Fig.42

ambiguïté sur quel dispositif il doit intervenir. Enfin, quand le logiciel attend une valeur, c'est par la rotation du codeur rotatif qu'il faut indiquer la donnée au programme. Par exemple sur la Fig.42 le programme attend la valeur de la Transition à partir de laquelle on désire **MODIFIER** le programme en cours de saisie. C'est donc le codeur rotatif qu'il faut actionner. Puis, la LED verte étant

éteinte, c'est avec le B.P.C. que l'on valide la valeur. Sur la Fig.43 le menu **EEPROM** est en cours d'exploration. Aucune LED n'est allumée, donc c'est le B.P.C. qui déclenchera la suite. On remarque que **>>> ÉTENDU <<<** avertit l'opérateur que c'est l'algorithme à 20 transitions qui est actuellement en mémoire. Le clic sur le B.P.C. fait passer à l'écran de la Fig.44 et comme le logiciel attend une réponse de type OUI ou NON au clavier, la LED verte clignote rapidement. C'est la touche du haut qui

est choisie pour valider, car c'est elle qui engendrera le plus de modification sur l'état de la machine. Toutes les autres touches seront interprétées comme un NON. Si l'on valide à ce stade, on écrase le programme type **ÉTENDU** actuellement préservé en EEPROM. Comme le perdre par inattention demanderait une réécriture notable, il y a une demande

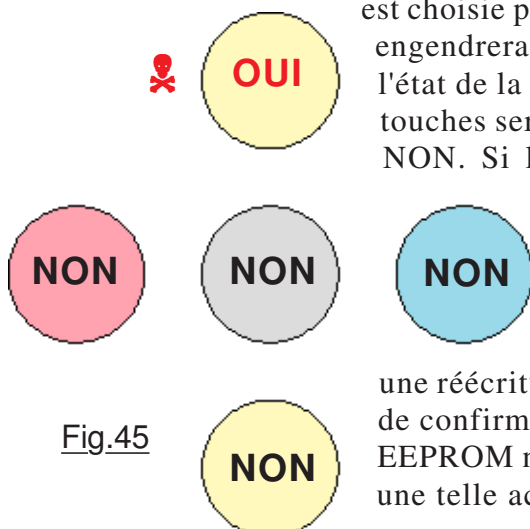


Fig.45

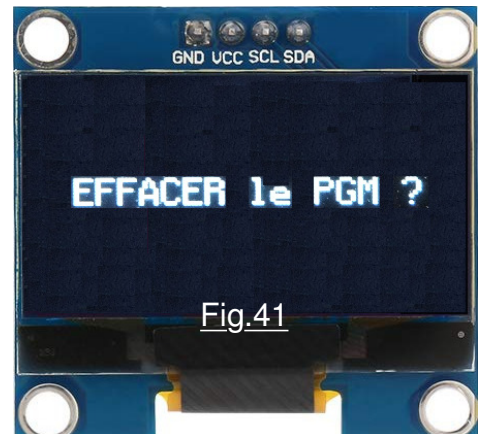


Fig.41

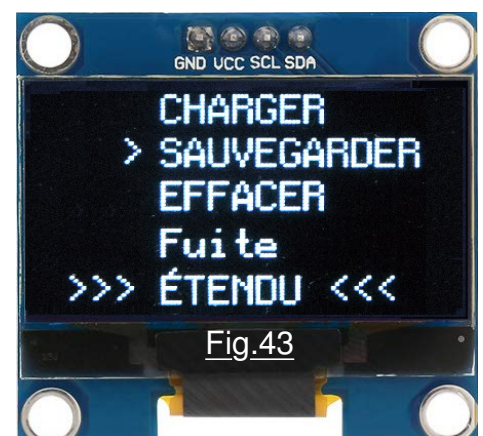


Fig.43

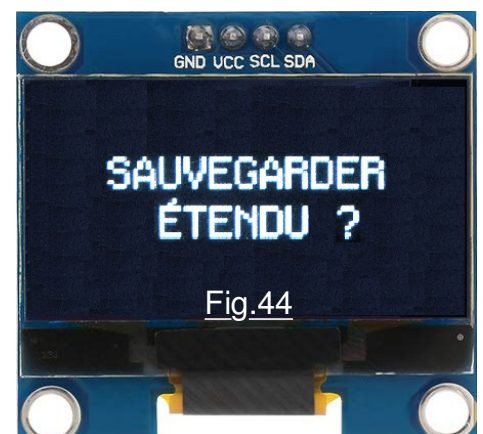
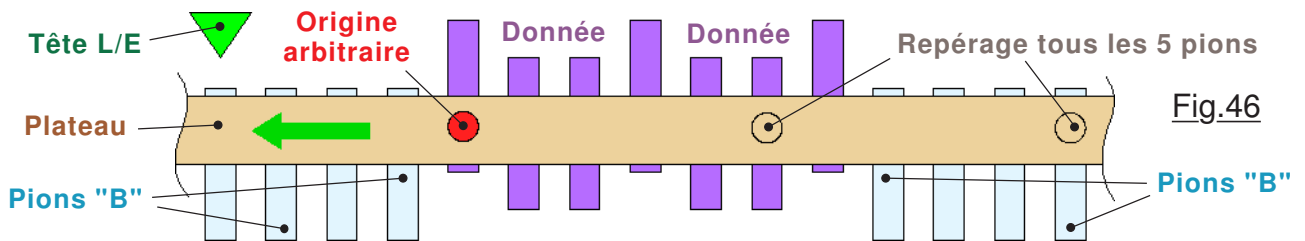


Fig.44

de confirmation. Noter au passage qu'effacer le programme **ÉTENDU** en EEPROM ne présente aucun intérêt. Pour simplifier le logiciel, si on tente une telle action, seulement trois BIPs sonores seront générés.

➤ Un standard personnel.

D'une façon générale, chaque fois qu'un programme fonctionne et qu'il reste au moins une transition disponible, j'ajoute en tête de grille une transition pour pouvoir placer les données à droite de la Tête de L/E et ainsi dégager les pions initiaux pour les avoir bien en vue sur la machine électromécanique quand on déclenche l'exécution avec **RUN**. Sur la Fig.46 qui représente



la zone avant du carrousel, la tête de L/E est symbolisée par le triangle vert. La transition que l'on doit placer en tête de l'algorithme consiste à faire tourner le plateau à gauche tant qu'un état "B" est détecté par la tête de lecture. Dès qu'un "0" ou un "1" est lu, on saute alors à la suite du programme. Dans ce but il faut décaler tout le programme d'une transition vers "le bas" pour pouvoir sur la transition n°1 installer les instructions de déplacement de la tête. Le problème, c'est que toutes les instructions de type **Trn** doivent être incrémentées. C'est tellement rébarbatif à faire que le logiciel

de la carte Arduino NANO prévoit la fonction **FORMATER** dans le menu **PROGRAMME** pour réaliser cette transposition de façon totalement automatique.

Fig.47

Ligne 01	Tr1-1 : . . 02 .
Ligne 02	Tr1-0 : . . 02 .
Ligne 03	Tr1-B : . G . .
Ligne 04	Tr2-1 :
Ligne 05	Tr2-0 : 1 G 04 .
Ligne 06	Tr2-B :
Ligne 07	Tr3-1 :
Ligne 08	Tr3-0 :
Ligne 09	Tr3-B :
Ligne 10	Tr4-1 :
Ligne 11	Tr4-0 :
Ligne 12	Tr4-B :
Ligne 13	Tr5-1 :
Ligne 14	Tr5-0 : 1 G 07 .
Ligne 15	Tr5-B :
Ligne 16	Tr6-1 :
Ligne 17	Tr6-0 :
Ligne 18	Tr6-B :
Ligne 19	Tr7-1 :
Ligne 20	Tr7-0 :
Ligne 21	Tr7-B :
Ligne 22	Tr8-1 :
Ligne 23	Tr8-0 : 1 G . . F
Ligne 24	Tr8-B :

Considérons la Fig.47 qui résume le travail que doit accomplir cette option à notre place. En première action, elle doit déplacer l'ensemble de la "grille de trous" d'une transition vers le bas, à condition toutefois qu'il reste au moins trois lignes de disponibles dans **Tr11** ou **Tr20** si le mode **ÉTENDU** est actif. Cette première étape est coloriée en vert pastel sur la Fig.47 montrant le "haut" du listage. Puis, la transition **Tr1** est remplacée automatiquement par le code du cadre colorié en rouge. Comme le logiciel ne peut pas savoir si le pion de gauche de la donnée sera un "1" ou un "0", arbitrairement il y aura saut à **Tr2** dans les deux cas et *il sera à la charge de l'opérateur de supprimer l'un des deux sauts dans l'algorithme si nécessaire*. La Fig.47 est un schéma pour visualiser ce que donnerait la fonction **FORMATER** si la troisième étape n'avait pas été prévue. Les instructions mises en

évidence en bleu programmées dans l'original pointeront une transition pas assez loin. La dernière étape que doit réaliser le logiciel consiste alors à incrémenter toutes les instructions **Trn** sauf la première naturellement. Quand avec l'écran de la Fig.48 on clique sur le B.P.C. le démonstrateur demande en Fig.49 confirmation. Car valider va changer les valeurs de toutes les transitions, alors il est souhaitable de ne pas déclencher cette fonction par inadvertance. Si on valide, la première analyse qu'effectue le programme c'est de vérifier qu'il reste au moins une transition de disponible en fin d'algorithme pour pouvoir décaler les instructions "vers le bas". Si l'intégralité de la grille est occupée, le verdict tombe en Fig.50 accompagné d'un avertissement sonore et a LED verte clignote rapidement.

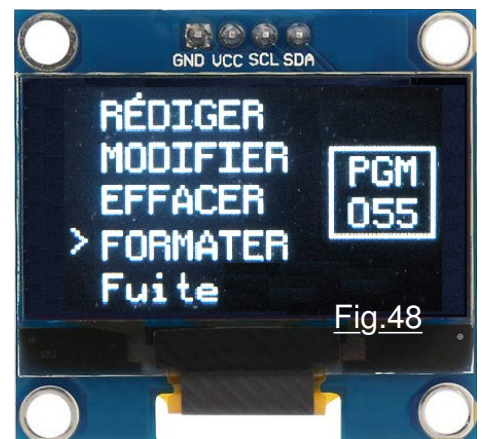




Fig.50

Lorsque l'opérateur accuse réception sur le petit clavier, il y a retour au **MENU de BASE**. Si la transposition est possible et que l'on n'est pas en mode Machine Étendue, le programme se contente d'effectuer les traitements décrit ci-avant et l'on revient directement au **MENU de BASE**. Par rapport au programme de **Machine Élémentaire**, l'approche a été simplifiée :

➤ **Changement de stratégie.**

Comprendre la nouvelle structure du programme exige un petit retour en arrière. Sur le programme qui anime la **Machine Élémentaire**, si l'algorithme est à 20 transitions, il faut au préalable recoder les instructions de type "F". La valeur de

transition **12** symbolise l'instruction de fin d'algorithme dans le cas standard alors que sur la Machine Étendue le code est **21**. Pour symboliser "F" maintenant on utilise uniquement **21**. Cette optimisation peut sembler anodine, mais elle fait économiser 174 octets de programme ce qui n'est pas du tout négligeable. De plus, *le protocole d'utilisation de la machine se simplifie* ce qui constitue un bénéfice collatéral tout à fait séduisant. Puisque nous en sommes à l'étude détaillée du logiciel, il me semble opportun d'ouvrir une parenthèse pour compléter les explications relatives à la Fig.39 qui présente l'occupation de la mémoire EEPROM. Pour se repérer dans ce listage mémoire, les débuts de chaque algorithme sont mis en évidence par les encadrement marrons du type **36**. Pour faire le lien avec la Fig.51, le tableau de la Fig.52 présente la traduction des **Références Décimales** en **Hexadécimal**.

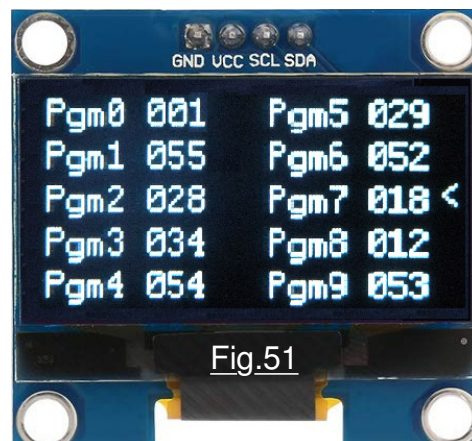


Fig.51

Décimal	46	55	28	34	54	29	52	18	12	53	21
Hexadécimal	2E	37	1C	22	36	1D	34	12	0C	35	15

Fig.52

Dans cette copie d'écran de la Fig.39 les cellules qui contiennent une donnée de type "F" sont mises en évidence par des encadrements rouges **15**. Maintenant que les deux menus **EEPROM** et **PROGRAMME** sont entièrement implémentés. Un petit bilan s'impose. La compilation avec la version 1.8.0 de l'**IDE** indique 14376 octets de programme *soit 46% de l'espace mémoire occupé*. Quand aux données, elles s'octroient 438 octet et 21% de la RAM dynamique. C'est finalement une bonne nouvelle. On pourrait s'inquiéter de constater que pas très loin de la moitié de la place disponible pour le programme est déjà consommée, alors que les routines qui font fonctionner la machine virtuelle ne sont pas encore installées. Ces dernières ne gloutonnent qu'environ 6000 octets. Ce qui sera le plus boulimique, ce sera les implémentations graphiques du **BARILLET**. Enfin, les **OPTIONS** que l'on pourra ajouter seront directement fonction de la place restante. Aussi, il semble judicieux de poursuivre chronologiquement le développement par :

PLANNING

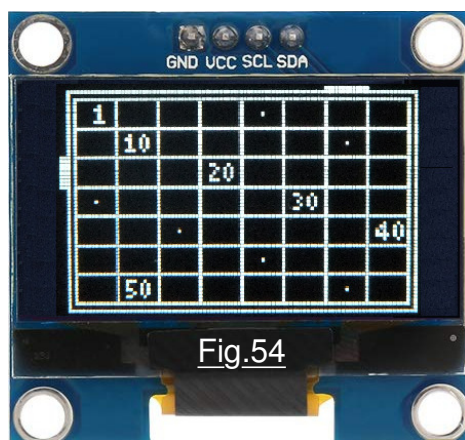
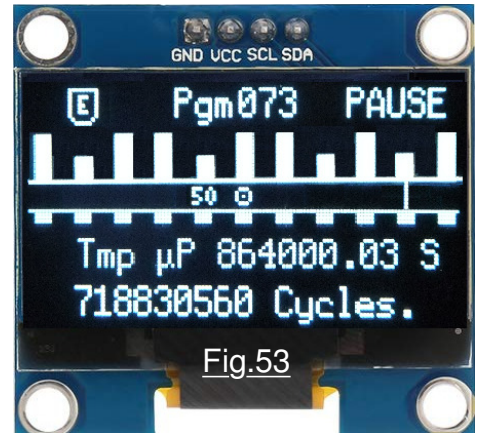
- Étude de la représentation du carrousel virtuel.
- Mise en place de la fonction RUN.
- Étude des options de représentation du BARILLET.
- Implémentation des OPTIONS envisageables.

➤ **Changer la Référence d'un algorithme.**

Intégrer dans le programme de la **Machine Élémentaire**, cette fonction n'est pas disponible sur la version autonome de la machine virtuelle. Elle encombrerait le menu **PROGRAMME** d'un item avec page écran supplémentaire pour une action qui à l'usage n'est pas très fréquente. Aussi, pour simplifier les protocoles et gagner une place significative dans le programme on se contentera d'utiliser la procédure suivante : L'algorithme étant chargé, on valide **RÉDIGER** dans le menu **PROGRAMME**, on précise la nouvelle **Référence** et on n'efface pas le programme. **Page 17**

06) La représentation du carrousel sur l'écran OLED.

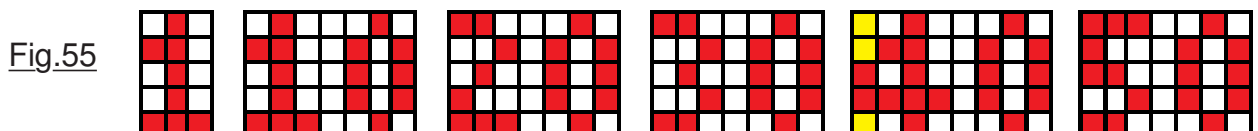
C'est la pierre d'achoppement qui va compliquer un peu l'agencement de ce projet. La difficulté réside dans l'utilisation d'un écran dont la définition est très limitée et qui n'a rien à voir avec celle du moniteur vidéo du P.C. de bureau. On pouvait dans la fenêtre du Moniteur de l'IDE afficher des lignes de textes presque sans limitation du nombre de caractères. Sur l'afficheur OLED avec la police utilisée, on peut au maximum afficher 21 caractères par lignes et cinq lignes par écran. Il va falloir se débrouiller avec cette limite et également optimiser les affichages pour avoir un rafraichissement pas trop lent. Question : Monter le BARILLET avec du texte ? Ou avec un graphique ? C'est précisément l'objet de cette étude qui va servir avec **P11_Afficher_Etat_du_BARILLET.ino** à trouver des représentations aussi simples à interpréter que possible. Ce démonstrateur ne traite pas d'un BARILLET implanté dans le programme. Il ne fait que simuler les affichages. Toutefois, les procédures ont été développées en fonction de l'implémentation binaire qui sera réservée au barillet. Les routines d'affichage sont



paramétrées pour pouvoir directement servir dans le programme complet. Le premier écran envisagé et montré sur la Fig.53 est graphique. Pour obtenir sur la page écran un dessin presque lisible, seule la zone qui sur la machine est dirigée vers l'opérateur est représentée et ne montre que onze pions. Il est évident que cette "lorgnette" ne sera pas très commode pour embrasser d'un coup les 56 BITS du plateau de la machine. Aussi une option sous forme de grille présentée sur la Fig.54 est indispensable. Les deux écrans sont bien remplis, je ne regrette vraiment pas d'avoir opté pour l'afficheur de 1,3 pouces de diagonale, et non l'équivalent OLED de 0,96 pouces que j'affectionne car il présente deux couleurs bleue et jaune ... mais il est si petit !

➤ **Police de caractères personnalisée.**

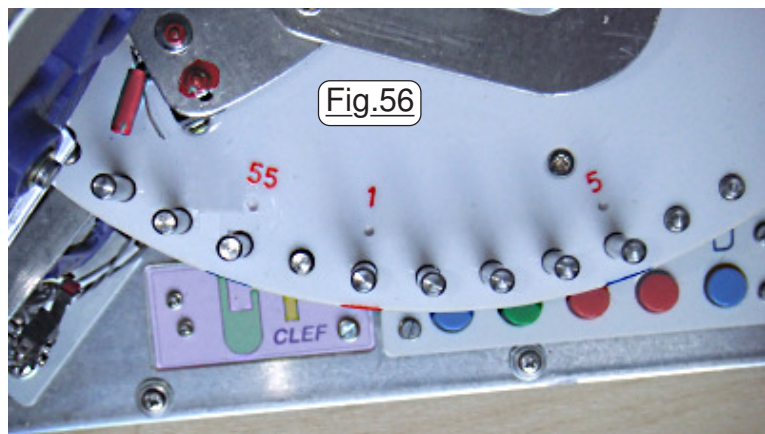
Celle qui a été choisie dans la bibliothèque **U8glib** présente une taille suffisante pour un visuel assez grand qui convient pour les personnes "du troisième âge" dont je fais partie. Lisibilité parfaite, avec pour inconvénient de limiter les textes sur une page écran à cinq lignes de vingt caractères. Il se trouve que la représentation du BARILLET impose à l'écran une définition bien plus élevée. Les textes dans la grille ou pour repérer les positions sur le carrousel doivent maigrir et devenir très petits. De ce fait, nous allons créer notre propre police de caractère présentée sur la Fig.55 qui se limite à six valeurs affichées.



Chaque chiffre s'inscrit dans une matrice de 3 PIXELs de large et de cinq PIXELs de haut. Les nombres sont séparés par une colonne d'un PIXEL ce qui entraîne pour les dizaines des mosaïques de 7 carrés de large par 5 de haut. La valeur 40 toutefois est un cas particulier. Le chiffre 4 pour rester lisible fait quatre PIXELs de large au lieu de trois. Il faudra en tenir compte pour le cadrage dans les cases de la grille, ou pour le centrage sous les pions.

➤ **Interprétation de la représentation graphique du carrousel.**

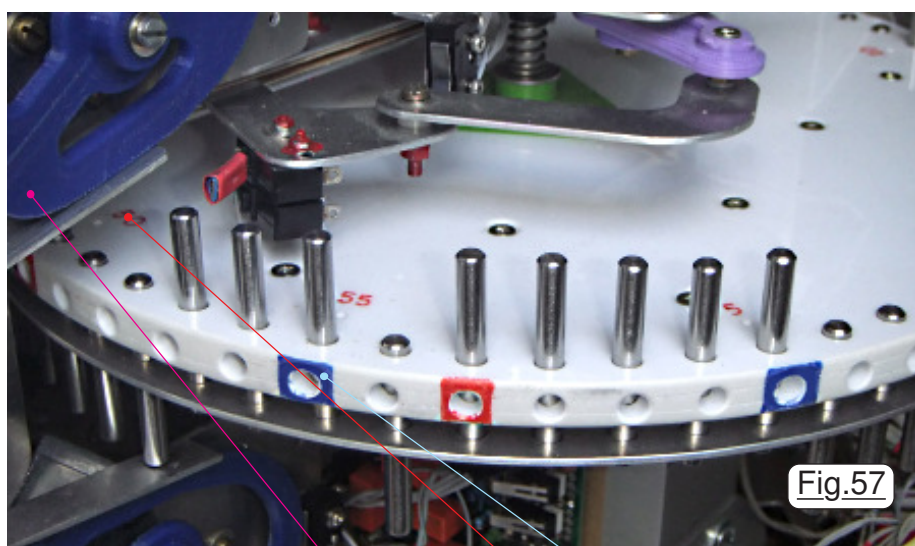
J'insiste sur le fait que le démonstrateur **P11_Afficher_Etat_du_BARILLET.ino** ne fait que présenter des possibilités et ne gère pas le BARILLET pour le moment. Beaucoup d'éléments sont prévus, mais tous ne seront peut être pas affichés. Par exemple le statut **PAUSE** semble séduisant, il restera toutefois à en tester la pertinence. Il en va de même pour le rappel de la référence de l'algorithme telle que **Pgm073** ne sera pas forcément conservée dans la version




ultime du logiciel. Nous allons détailler la représentation "fenêtrée" du plateau de la machine virtuelle. Pour établir le lien avec le plateau du prototype électromécanique, il nous faut absolument établir le parallèle entre la réalité matérielle, et la symbolique adoptée pour sa représentation sur l'écran de l'ordinateur. *(Ce chapitre reprend celui proposé dans le didacticiel sur La Machine Élémentaire. Comme ce tutoriel constitue un tout, ne vous étonnez pas de retrouver ce "doublon".)* Commençons par regarder

Image 15.JPG et Image 16.JPG qui sont préservées dans le dossier <Galerie d'Images>. Ces deux photographies présentent le BARILLET du prototype en cours de réalisation. Il importe sur

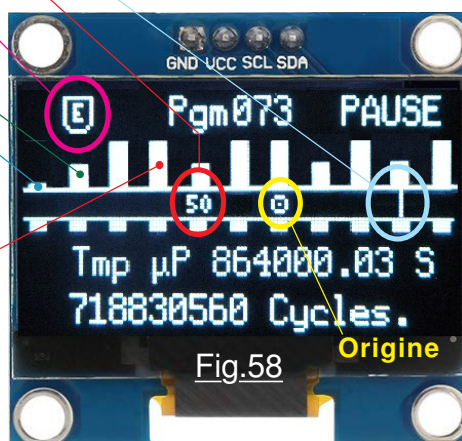
ces deux exemples de voir comment sont repérées les positions des pions de cinq en cinq. Sur ces images vous constaterez que les BITS ont été repérés et numérotés en rouge tous les cinq pions sur le dessus du plateau. Ils sont également repérés en rouge et en bleu comme visible sur la Fig.57 également tous les cinq pions sur la tranche du carrousel. *Ce ne sont que des repères visuels utiles à l'opérateur pour analyser plus aisément le résultat du déroulement d'un algorithme.*




Sur la Fig.58 on repère facilement le symbole rose de la Tête d'Écriture. Que ce soit sur la représentation graphique ou dans la GRILLE seul le 1 et les dizaines sont représentés par les chiffres 1, 10, 20, 30, 40 et 50. Les valeurs avec le chiffre 5 sont représentées par la barre verticale tracée sur le Plateau virtuel. *(Dans le médaillon bleu clair sur la Fig.58.)* Dans la GRILLE ces valeurs "en cinq" sont mises en évidence par un PIXEL central dans la case concernée. L'Origine arbitraire, positionnée ici en 52 sera mise en évidence par le petit dessin . La représentation de l'origine sera prioritaire sur celle des autres repères du plateau.

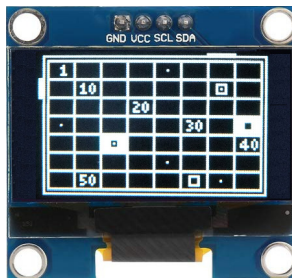
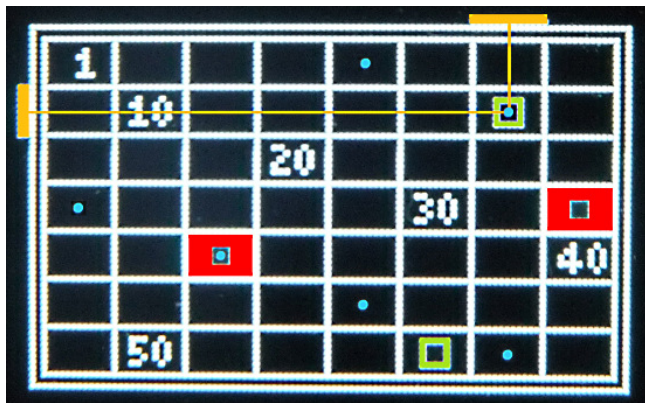
Tête Ecriture

Pion à "0"
Pion à "B"
Plateau
Pion à "1"



➤ Visualisation complète du BARILLET.

Représenter simultanément les 56 pions du carrousel s'avèrera utile pour examiner le résultat d'un traitement intervenant sur un grand nombre de BITS. Ce sera indispensable pour faciliter la saisie d'un état initial avant de pouvoir activer un algorithme en automatique avec RUN. Le nombre de 56 pions est idéal pour les représenter sous forme d'octets, c'est à dire un damier de 8 x 7. Le dessin de la Fig.59 qui surcharge une photographie va nous servir à préciser le mode de représentation. Les six positions "en 5" sont repérées par un pixel central qui a été surchargé par un disque tracé en bleu clair. La représentation des pions à "0" et à "1" sera prioritaire sur celle des repères 1, 10, 20, 30, 40 et 50. Si la case est vide ou ne contient que le repère de position sur le plateau, c'est que l'état du pion concerné est à "B". Les états "0" sont indiqués par le symbole  qui est un petit carré centré dans la cellule. On peut remarquer que cette représentation n'occulte pas le PIXEL central qui



repère les positions "en 5". Par exemple les BITS n°15 et n°54 sont à l'état "0". Repéré en rouge sur la Fig.59 un BIT à "1" sera indiqué en remplissant entièrement la cellule tout en ménageant le PIXEL central. Dans notre

exemple le BIT n°35 est à l'état "1". Le PIXEL au milieu confirme la valeur 35. Pour éviter de surcharger le contenu d'une cellule quel que soit

Fig.59 l'état de son BIT, l'Origine est repérée par les deux index surchargés en orange sur la photographie prise en exemple. La cellule concernée se trouve à la "croisée" des indexages. Dans l'exemple de la Fig.59 on peut affirmer que l'Origine arbitraire est en position 15 sur la machine.

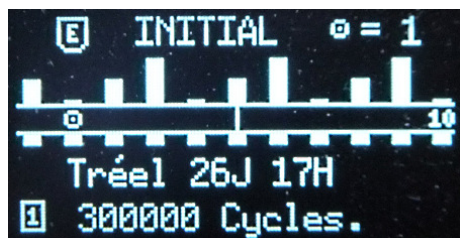
➤ Durée maximale de fonctionnement.

L'affichage de la Fig.58 indiquant des durées et le nombre de cycles d'HORLOGE effectué va présenter d'autant plus de caractères à visualiser que les valeurs seront importantes. Par ailleurs, pour visionner ces données il faut imposer une mise en page. Si l'on présente les valeurs trop à gauche, l'ensemble ne sera pas "équilibré". Au contraire, trop à droite des caractères vont être hors de la mosaïque. Il importe donc d'établir un compromis entre positionnement latéral et ne pas risquer le débordement. Pour ne pas trop "brider" la machine, *j'ai arbitrairement décidé que l'utilisateur ne laissera fonctionner son appareil qu'au maximum dix jours d'affilée*. Quelques petits calculs s'imposent :

- Quand l'algorithme se déroule en "mode aveugle", la cadence de traitement se fait à vitesse maximale. Dans ces conditions l'ATmega328 déroule environ 4400 tours d'HORLOGE par seconde.
- Si on laisse fonctionner la machine virtuelle durant dix jours sans interruption, l'HORLOGE aura effectué : $86400 \times 10 \times 4400$ Cycles soit l'affichage **380160000 Cycles** d'où le positionnement du texte adopté sur la Fig.58 qui assure un non débordement des limites de l'afficheur.
- Le temps mesuré du fonctionnement du microcontrôleur sera de $86400 \times 10 \times 1000$ car il est calculé en millisecondes. On n'affichera que les centièmes de secondes ce qui conduira à la valeur la plus grande possible de **864000.00 S** valeur simulée sur la page écran photographiée.

➤ Durée Machine en temps réel maximal.

Problème strictement analogue, dans la version complète, si c'est possible on aura en option la possibilité d'afficher le temps que prendrait la machine électromécanique. Sur un fonctionnement simulé de dix jours, en supposant que les 33 lignes du programme contiennent l'écriture d'un "1" qui est la plus longue, ("1" : 1,8S) une rotation qui exige 0,27S et un saut de 10 transitions, le cycle HORLOGE le plus lent consomme environ 7,7S. Dans ces conditions pour les 380160000 Cycles la mécanique devrait fonctionner durant 29.272.320.000 secondes soit environ 928 années



et 80 Jours. (OUF!) Aussi, compte tenu du fait qu'une ligne affichée ne peut contenir que 20 ou 21

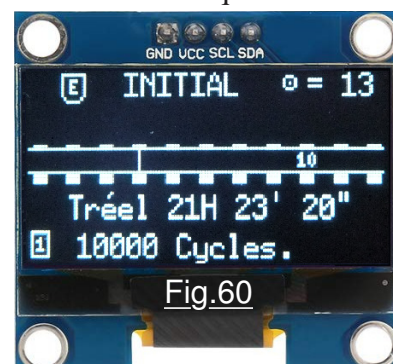
Fig.61 caractères au maximum, la

durée sera affichée à la place de la donnée mesurée du temps microcontrôleur **Tmp µP 864000.03** sous trois formes envisageables (Fi.60, 61 et Fig.62) :

- Durée importante : **T réel 280A 288J.**
- Durée moyenne : **T réel 26J 17H.**
- Durée "raisonnable" : **T réel 21H 23' 20".**

Il ne reste plus qu'à intégrer ces affichages dans le programme complet, ce sera l'objet du prochain démonstrateur.

Fig.62



07) Intégration de la gestion du BARILLET dans le logiciel.

Outre l'ajout des routines de visualisation du carrousel sous forme graphique ou par la grille de la Fig.59 il faut développer les fonctions relatives au menu **BARILLET** et introduire les données qui représentent les 56 pions du plateau de la machine virtuelles, et celle des 56 BITS correspondant à l'état initial. C'est le démonstrateur **P12_Gestion_du_BARILLET.ino** qui gloutonne le plus de place en mémoire de programme, car il devient très "bavard" sur les nouveaux textes affichés et surtout sur les pages d'écran très encombrées comme nous allons le constater. Notons au passage que les copies d'écran Fig.60, Fig.61 et Fig.62 sont issues d'un réel calcul issu de la durée d'un traitement. Seule cette durée est simulée, la routine de détermination et d'affichage du temps réel que prendrait la machine électromécanique sont codés pour le programme futur.

➤ Un menu pour **BARILLET** modifié et optimisé.

P hénomène d'une banalité qui confine au pléonasme, le développement d'un programme fait forcément émerger des faiblesses et des décisions pénalisantes qui impliquent de restructurer régulièrement des approches qui semblaient taillées dans le marbre. Pour mémoire, l'image ci-dessous reprend la Fig.13 qui montrait les items du menu **BARILLET**. Comme on peut le constater sur la

Fig.63 qui montre la nouvelle architecture de cette page écran, les fonctions qui simulent la rotation à gauche et à droite du plateau ont été supprimées. En effet, avoir à invoquer ce menu chaque fois que l'on désire faire tourner le carrousel s'est rapidement montré totalement inefficace. Dans la conception précédente, on pouvait avec **Affiche** montrer l'état du **BARILLET** initial, et avec **PLATEAU** celui de la machine suite à un

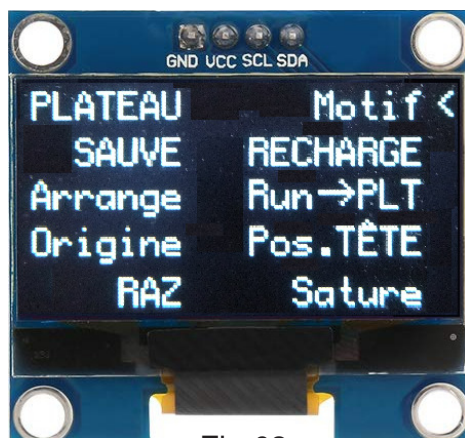


Fig.63

traitement **RUN**. L'expérience montre sans contestation que l'on doit pouvoir passer de l'un à l'autre rapidement. Dans la nouvelle mouture l'item **Affiche** est donc intégré directement dans l'item **PLATEAU** symbolisé par la flèche verte, fonction plus complète qui prend la première place dans le menu, ce que suggère la flèche orange. Du coup on ouvre la possibilité de trois options nouvelles, et sur la Fig.63 le texte **Init.BIT** a été remplacé par **Sature** à mon sens plus explicite. Nous allons point par point analyser toutes les fonctions actuellement disponibles dans le menu **BARILLET**.

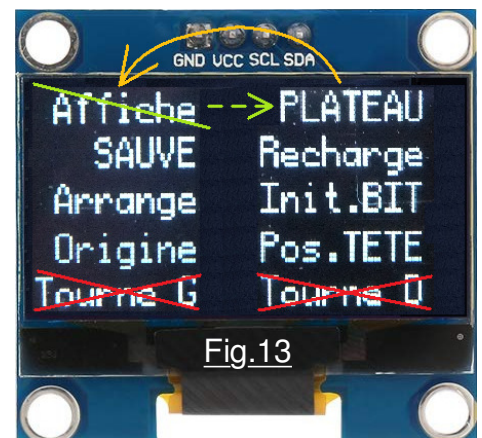


Fig.13

➤ La fonction **PLATEAU**.

P remière dans la liste des items du menu **BARILLET**, cette fonction est de loin l'une des plus délicate à avoir été développée, car la représentation du carrousel s'est avérée infiniment plus délicate à créer et à déverminer qu'une évaluation initiale naïve ne le laissait présager. Noter au passage que les informations **Pgm073 PAUSE** montrées sur la Fig.58 par exemple ne concernent que le mode **RUN** qui pour le moment n'est toujours pas développé et ne concernent pas le protocole **PLATEAU**.

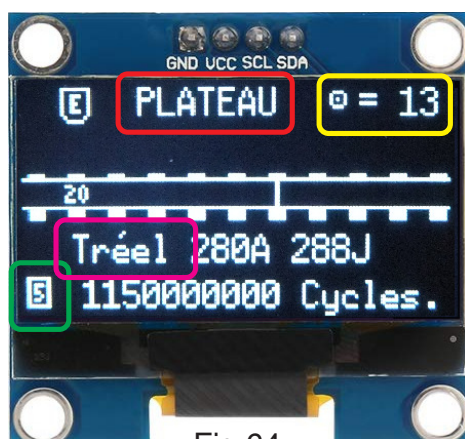


Fig.64

FENETRE / GRILLE

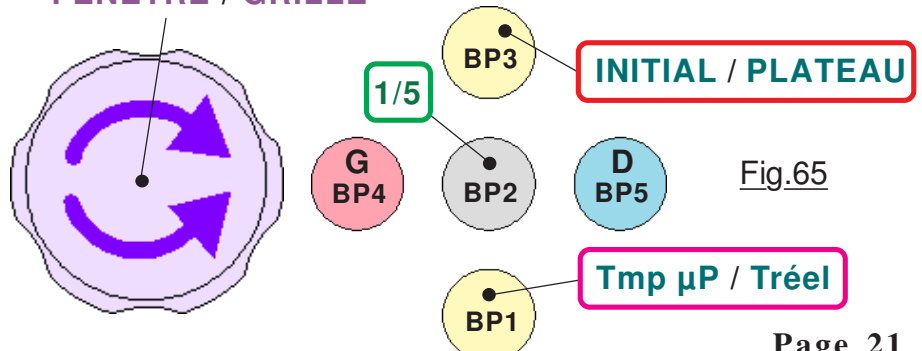


Fig.65

Note relative au démonstrateur P12.

Lisant les explications relatives à **P12_Gestion_du_BARILLET.ino**, vous allez forcément tester les divers items du menu **PLATEAU** pour concrétiser les thèmes de ce chapitre. Aussi, il importe de savoir que chaque fois que vous allez cliquer sur l'un des **BP** du petit clavier représenté en Fig.65 pour en observer les effets, la valeur du nombre de **Cycles** supposés réalisés par l'HORLOGE virtuelle est augmenté de 50.000.000 pour visualiser les conséquences sur **Tmp µP** et sur **Tréel**. La valeur de l'incrément est choisie très importante pour arriver en quelques clics à des durée très importantes et simuler jusqu'à 10 jours de fonctionnement ininterrompu. La fonction **RUN** actuelle permet à convenance de replacer le nombre de **Cycles** à zéro. Pour tester une augmentation moins importante à chaque **BP** il suffit de modifier la valeur de l'incrément dans la procédure **Affiche_le_BARILLET()**. Enfin, comme précisé en tête du démonstrateur, la procédure provisoire **RUN** fait un BIP et charge **PLATEAU** à moitié avec des "0" et l'autre moitié avec des "1". (Voir la Fig.74 en page 26) Cette configuration est commode pour expérimenter les rotations **G** et **D** et utile également pour tester la fonction **Run->PLT**.

Considérons la Fig.64 qui présente le carrousel sous l'aspect **FENETRE**, c'est à dire avec la symbolique de la Fig.58 donnée en page 19. Sur cet exemple on montre la configuration du **PLATEAU**, c'est à dire avec les effets qui seraient engendrés par l'activation d'un algorithme. Avec un clic sur le **BP3** peut à tout moment visualiser l'état **INITIAL** suite à l'utilisation de la fonction **Arrange**. Comme on peut s'en douter, le **BP4** fait tourner le plateau à **Gauche** alors que **BP5** le déplace à **Droite**. Si l'on observe le petit encadré vert clair, la valeur **5** est affichée. En cliquant sur le **BP2** on fait alterner cette indication entre **1** et **5**. Cette information correspond au nombre de pions qui vont se déplacer sous la tête de L/E quand on fait tourner le carrousel. Il devient alors très facile de le faire tourner "lentement" ou "rapidement" pour visualiser à notre guise la zone du carrousel qui nous intéresse. Autant sur la représentation **FENETRE** on sait quel est le repère du pion qui se trouve sous la tête de L/E, le n°20 sur la Fig.64, autant le pion qui a été choisi pour **Origine** arbitraire peut se trouver hors des limites affichées. (C'est le cas dans cet exemple.) Aussi, dans l'encadré jaune, la valeur de

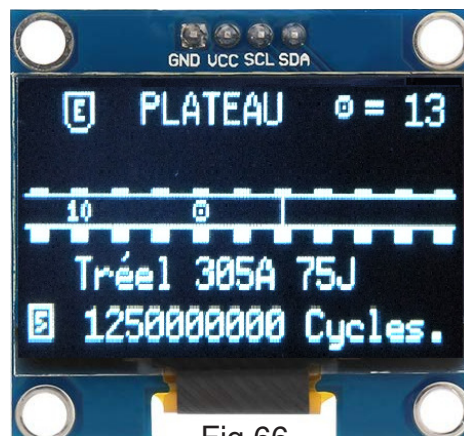


Fig.66

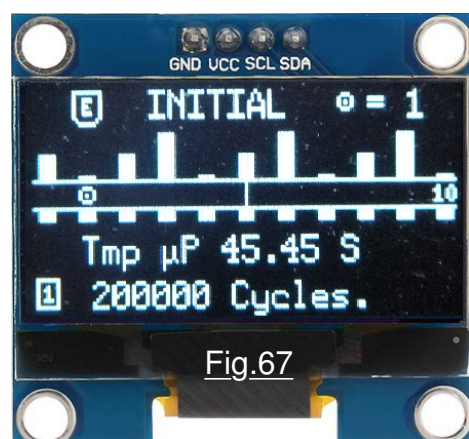




Fig.67

l'**Origine** symbolisée par  est précisée numériquement. Avec **BP1** on alterne l'affichage entre la valeur de **Tmp µP** et celle de **Tréel**. Sur la Fig.66 l'origine se trouve *hors des repères "numériques"* de position des pions sur le plateau. Par contre, sur la Fig.67 c'est le BIT n°1 qui se trouve sous la tête de L/E. Quand il y a superposition entre un repère "numérique" (Dont les intermédiaires "5" repérés par le trait vertical.) et la valeur de l'**Origine**, c'est le symbole  qui est *prioritaire*, le "numérique" étant ignoré. Comme pour les autres menus, cliquer sur le B.P.C. fait sortir de la représentation du **PLATEAU** pour revenir au **Menu de BASE**. Toutefois, avant de sortir de ce mode, tourner le bouton du codeur rotatif. Quel que soit le sens de rotation, on alterne la visualisation entre l'option graphique du carrousel et la représentation **GRILLE** qui sous la forme d'un damier présente la totalité des 56 pions. Cette page écran est gavée d'informations qu'il importe de détailler. On retrouve globalement la présentation donnée en Fig.54 de la page 18 avec de nouvelles informations complémentaires. Que l'on soit en mode **FENETRE** ou en représentation de type **GRILLE** montré sur la Fig.68, comme vous pouvez l'observer sur la Fig.69 l'utilisation du petit clavier et du codeur rotatif est strictement la même. Seule petite différence, en option d'affichage de type **GRILLE** le **BP1** est sans effet car l'affichage des durée n'est plus pertinent, et se contente de faire un BIP sonore pour en avertir l'opérateur.

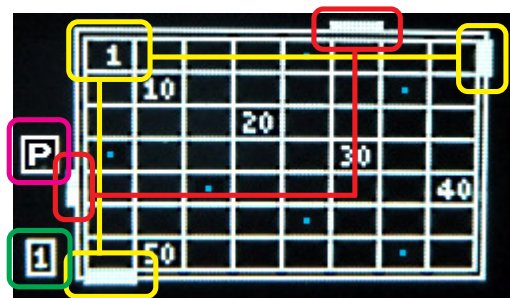


Fig.68

FENETRE / GRILLE

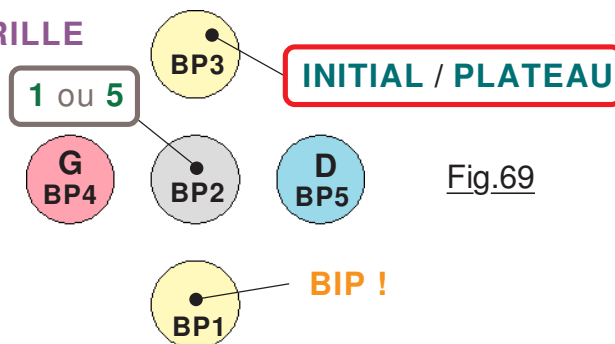
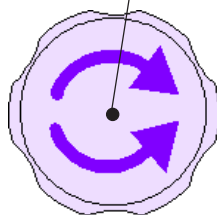



Fig.69

Exemple proposé suite à un RESET, sur la Fig.68 le damier est "vide" car tous les pions sont forcés à l'état "B". De ce fait, tous les repérages sont visibles avec les symboles numériques pour les positions de 10 en 10 et par les petits carrés centraux coloriés en bleu pour les positions des repères "en 5". Quand on clique sur **BP3** on alterne en configuration **PLATEAU** indiqué par le **P** dans l'encadré rose, ou en affichage **INITIAL** représenté par  à la place du **P**. En coordonnées cartésiennes, la "croisée" des curseurs "rouges" indique la position de la tête de L/E. Dans l'exemple c'est le pion n°38 qui est sous la tête de L/E. La position de l'**Origine** est précisée par le curseur situé sur le coté droit de l'encadrement et par celui qui est sur le coté bas de la grille. Dans cet exemple l'**Origine** arbitraire est placée sur le pion n°1. (*Surcharges en jaune.*) Le nombre de pas effectués quand on clique sur **BP4** ou **BP5** est indiqué dans l'encadré vert.

ATTENTION : En mode **GRILLE** le curseur se déplace à droite quand on clique sur **G** et à gauche quand on utilise **D**. C'est normal, car *dans cette représentation le carrousel est immobile, on voit donc se déplacer la tête de L/E, et son mouvement relatif est inversé.*

➤ RECHARGER ou SAUVEgarder une configuration BARILLET.

Lorsque l'on initialise la mémoire non volatile de l'ATmega328 avec l'utilitaire **P00_Initialiser_EEPROM.ino** on y inscrit une configuration carrousel qui ne contient que des "0" et des "1" représenté sur la Fig.70 qui se traduit par un visuel très particulier. Quand on **SAUVE** une configuration on inscrit la valeur des 56 BITS du carrousel initial, ainsi que la valeur de l'**Origine** et celle de la position de la tête de L/E. En EEPROM l'origine est initialisée à 3 et la tête de L/E à 33. L'inconvénient du tableau de la Fig.70 est que tous les jalons numériques sont cachés, il ne reste que les points centraux des "valeurs en 5", d'où un repérage plus délicat. L'utilisation de l'item **RECHARGE** commence par afficher **CHARGER OUI** vers le centre de l'écran. Si l'on fait tourner le codeur rotatif, la valeur **OUI** alterne avec **NON**. On termine l'action positive ou négative en cliquant sur le bouton central du codeur rotatif et si **OUI** est validé le rechargement se fait inconditionnellement quelle que soient les données en EEPROM. Si le transfert est validé, la **GRILLE** de la Fig.70 est affichée et l'on peut vérifier la configuration des pions, la position de la tête de L/E ainsi que le pion choisi pour l'**Origine**. La LED triple clignote en vert signalant à l'opérateur qu'il doit cliquer sur le petit clavier pour revenir au **MENU de BASE**. De façon tout à fait analogue, cliquer sur **SAUVE** fait afficher **SAUVEGARDER OUI** que l'on modifie avec le codeur rotatif et que l'on valide avec le B.P.C. Quel que soit l'état **OUI** ou **NON** il y a retour au **MENU de BASE** sans autre information.

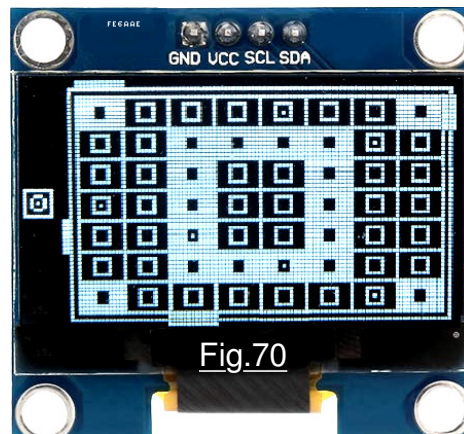


Fig.70

➤ La fonction Arrange.

L'activation de cette fonction d'*initialisation manuelle du BARILLET* ouvre un écran qui ressemble étrangement à celui de la Fig.70 avec toutefois un petit carré en bas à gauche qui contiendra la valeur **1** ou la valeur **5**. On se doute que cette information est relative au nombre de pas qui seront effectués quand on va se déplacer dans la grille. Diverses expérimentations ont été testées, car il fallait aboutir à une procédure agréable. Initialement, le codeur rotatif faisait alterner l'inscription d'un "B", d'un "0" et d'un "1" dans la cellule concernée, et les déplacements

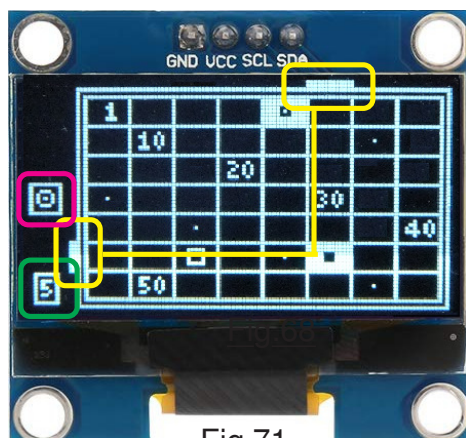


Fig.71

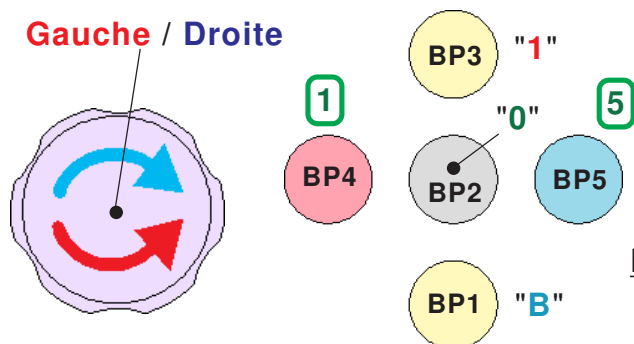


Fig.72

étaient gérés au clavier. Au final, c'est un protocole inverse qui a été implémenté car il s'est avéré bien plus convivial lors des essais. Dans la version actuelle du démonstrateur, le codeur rotatif comme l'indique la Fig.72 déplace les curseurs qui indiquent en coordonnées cartésiennes la cellule en cours de saisie. Dans notre exemple les curseurs sont encadrés en jaune et pointent le pion n°46. Pour rappel qu'il s'agit du carrousel **INITIAL**, le petit carré repéré en rose affiche le symbole de l'**Origine** par opposition au **P** du **PLATEAU**. La répartition des touches est verticale pour l'inscription d'un **"B"**, d'un **"0"** et d'un **"1"** et le pas est égal à **1** pour le BP de gauche et de **5** pour celui de droite.

➤ La fonction **Origine**.

Poursuivons dans l'ordre l'exploration des items du menu **BARILLET**. On trouve la possibilité de choisir la valeur arbitraire du pion défini comme **Origine**. Le centre de l'écran affiche le texte **Origine en : 13** qui à l'ouverture propose la valeur **1**. En tournant le codeur rotatif on incrémente ou on décrément la valeur avec une borne inférieure égale à **0** qui alterne avec **1**. Dans le sens positif, **56** fait recycler à **1**. Comme l'**Origine** peut être choisie n'importe où sur le **BARILLET**, on doit pouvoir se déplacer rapidement. C'est le clavier qui octroie la faculté de choisir des pas unitaires de 1, 5 ou de 10 ce que précise la Fig.73 avec **BP2** et **BP3** qui ne sont pas utilisés et qui engendrent un BIP d'avertissement sonore. La sortie de la saisie se fait avec le B.P.C.

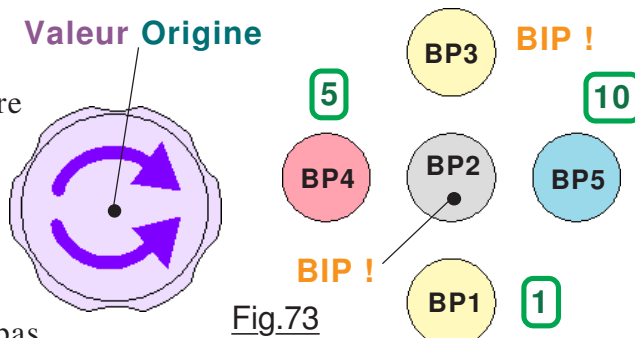


Fig.73

➤ La fonction **Pos.TÊTE** du menu **BARILLET**.

C'est le complément direct de la fonction précédente qui ouvre la saisie avec le texte **Position TÊTE : 33**. Comme pour le choix de la valeur de l'**Origine**, la valeur initiale est forcée à **1**. Le comportement du clavier et du codeur rotatif sont strictement identiques aux informations de la Fig.73 puisqu'ici aussi il s'agit de mentionner l'un des pions quelconques du **BARILLET**. De la même façon, cliquer sur le B.P.C. ramène au **MENU de BASE**.

➤ La fonction **RAZ** du menu **BARILLET**.

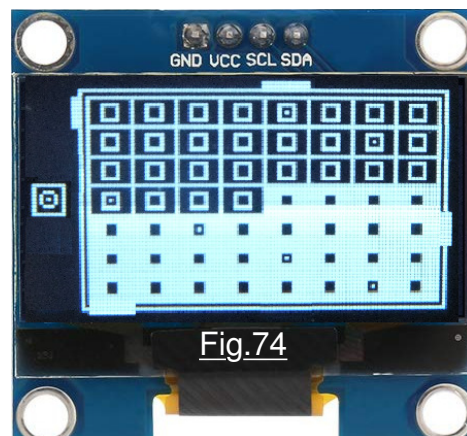
Signe qui signifie habituellement **Remise À Zéro**, dans notre cas l'implication est un peu différente. Si l'on valide cette option en cliquant sur le B.P.C, il y aura retour au **MENU de BASE** et tous les pions seront forcés à l'état **"B"**. (*État "B" et non "0"*.) Le petit clavier est ignoré. En tournant le codeur rotatif on alterne entre **OUI** et **NON**. Valider avec **NON** ramène directement au **MENU de BASE** alors que **OUI** fait afficher la **GRILLE** pour visualiser l'état du **BARILLET**. La LED triple clignote en vert pour inciter l'opérateur à cliquer sur l'une des touches du petit clavier.

➤ La fonction **Sature** du menu **BARILLET**.

Analogue à la fonction précédente, cette option est prévue pour remplir entièrement le barillet avec des **"B"**, des **"0"** ou des **"1"** à l'initiative de l'utilisateur. Le comportement est assez analogue à celui de **RAZ**. Le petit clavier est toujours ignoré. Le codeur rotatif fait alterner les valeurs potentielles **"B"**, **"0"**, **"1"** avec des affichages tel que **Tout le plateau à "B"** et surtout **Fuite** qui permet de revenir au **MENU de BASE** sans rien changer. Comme pour **RAZ** la sortie de cette fonction se fait avec le B.P.C. et affichage de la **GRILLE** si la validation est effective.

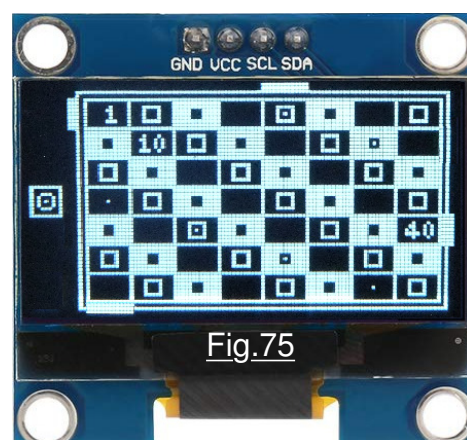
➤ La fonction **Run->PLT** du menu **BARILLET**.

Possibilité qui ne sera pas forcément la plus utilisée, cette procédure offre au programmeur la faculté de recopier le contenu du **PLATEAU** dans **INITIAL**. Pour mémoire, la valeur mémorisée dans **INITIAL** est purement virtuelle et n'existe pas sur la machine matérielle. Dans le cas de la machine virtuelle, contrairement au prototype électromécanique, on ne peut pas réactiver un **RUN** sur un carrousel qui contiendrait le résultat d'un traitement déjà engagé sur la machine. C'est dans ce cas qu'il devient bien commode de recopier intégralement la configuration de **PLATEAU** dans **INITIAL**, y compris la valeur de la position de la Tête de L/E. L'ouverture de cette fonction se fait sur le texte **PLT vers Initial OUI**. Le petit clavier est ignoré à ce stade et c'est le codeur rotatif qui fait alterner **OUI** et **NON**. Comme pour les fonctions précédentes, la sortie avec **NON** ramène directement au **MENU de BASE** alors que **OUI** fait afficher la **GRILLE** et la LED verte clignote. Pour que cette grille puisse afficher un état du plateau "issu d'un algorithme", en cliquant sur **RUN** on génère un plateau particulier d'un "visuel facile" dont la Fig.74 présente l'aspect. Noter que **RUN** génère un BIP sonore pour signaler qu'il a bien été pris en compte.



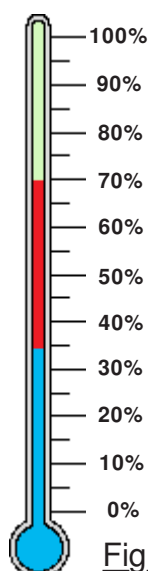
➤ La fonction **CHARGER le Motif** du menu **BARILLET**.

Dernier item à ne pas avoir été détaillé, c'est bien parce qu'un trou était présent dans la liste que ce petit "outil" a été ajouté au menu. Quand on valide alors que l'affichage de la page écran indique **CHARGER le Motif OUI** le carrousel va être entièrement rempli par une succession de série "**B 0 1**". Copiant les protocoles des fonctions précédentes, la **GRILLE** est affichée et présente l'aspect caractéristique de la Fig.75 avec toutes les cellules initialisées, mais qui laissent voir trois repères numériques. Cette configuration est très utile pour observer les affichages de **INITIAL** quand on fait tourner le barillet, effet visuel formateur, que l'on soit en mode **FENETRE** ou en visuel de **GRILLE**. C'est du reste avec cette configuration que la Fig.67 a été générée. L'affichage de la configuration du carrousel fait clignoter la LED triple en vert, incitant l'utilisateur à cliquer sur l'une des touches du clavier pour revenir au **MENU de BASE**. Noter que cette initialisation automatique de la position des 56 pions n'affecte que ces derniers et ne modifie ni la position de la Tête de L/E ni celle de l'**Origine**.



➤ Un bilan de consommation des ressources.

A n'en pas douter, l'implantation de toutes les fonctions du menu **BARILLET** a gloutonné un espace mémoire de programme très important. Avant de créer tout l'arsenal de gestion et d'affichage du carrousel, le programme consommait 34% de la zone réservée au logiciel. (*Compilateur 1.8.0*) Sur le "thermomètre" de la Fig.76 cette dépense initiale est coloriée en bleu clair. Le prix à payer pour ajouter les affichages et les fonctions du **BARILLET** est de 35% colorié en rouge dans le capillaire fictif. La consommation totale est de 69%. Il ne nous reste plus que la zone verte, soit moins du tiers de la zone programme, pour faire fonctionner réellement cette machine fictive, et la pourvoir du menu des **OPTIONS**. Et bien, contrairement à ce que vous risquez de penser, *cette zone verte me semble tout à fait suffisante*. En effet, les routines graphiques sont boulimiques en ressources. Elles sont presque toutes émulées. Quand au **RUN** proprement dit, à mon sens il consommera moins de 10%, car déjà le calcul des durée est effectif, faire tourner le **PLATEAU** est en place etc. Tout m'incite à croire qu'il y aura encore assez d'espace pour les **OPTIONS**. L'optimisation à outrance du code semble porter ses fruits et va continuer d'importance. On persévère et on signe ...



08) Ajout de la gestion du RUN dans le logiciel.

C'est forcément avec cette fonction que tout ce qui a été réalisé en amont va enfin prendre tout son sens. C'était un passage obligé, car on ne peut pas faire "tourner" un algorithme sans pouvoir au préalable le créer, puis configurer le plateau de la machine. Enfin, quand un programme est activé en automatique, il reste encore à pouvoir en observer les effets. Tout le développement effectué en amont avait pour but ultime de posséder tous ces éléments pour enfin rédiger "le moteur" de cette machine virtuelle et l'activer pour le meilleur ... ou pour le pire ! C'est le démonstrateur [P13_Introduction_du_RUN.ino](#) qui engrange toutes les routines du [RUN](#). Glouton en ressources de l'ATmega328 il consomme 12% de l'espace réservé au programme. Cette boulimie en code peut sembler exagérée. Il n'en est rien, car de nombreux écrans pour les options ont été intégrés, et pratiquement tout ce qui concerne [RUN](#) est actuellement implémenté.

➤ Un outil crédible.

Constitué de cinq "switch" le simulateur de clavier de la Fig.26 en page 9 est très utile pour dégrossir l'ossature du logiciel. Toutefois, arrive un moment où la structure définitive du clavier qui sera intégré dans la machine matérielle devient indispensable et ce pour deux raisons. La première réside dans sa géométrie. Les cinq touches de l'ensemble ultime seront placées en croix. Il importe de s'assurer que les choix qui sont faits dans les divers protocoles prouvent leur pertinence. Cette validation n'est fiable qu'avec la version définitive du petit clavier. La deuxième raison résulte du comportement des composants définitifs. Le choix actuel pour les boutons poussoir emploie des mini-composants dont le coût reste acceptable. Outre leur petite taille, ils sont munis de cabochons de toutes les couleurs offrant un bon éventail de combinaisons envisageables. On peut ainsi choisir les teintes à notre guise. Sur cette photographie en **A** se trouve le corps des ces petits boutons poussoir. La partie active qui s'enfonce quand on clique porte un tenon conique sur lequel s'emboîte le cabochon **B** disponible en sept couleurs. En **C** l'un de ces composants est assemblé. Ils se positionnent facilement sur une grille au pas standard. Sur la Fig.78 un clavier de démonstration fait désormais partie intégrante "du laboratoire". Cette réalisation a montré que la qualité électrique de ces composants reste "médiocre", c'est à dire que lorsque l'on clique, le passage au travail s'accompagne d'un grand nombre de rebonds. Du coup, la procédure de lecture a complètement été revue pour obtenir un fonctionnement antiparasite fiable. Cette dernière a été reportée sur tous les démonstrateurs qui précèdent. Avantage : Quels que soient les composants approvisionnés, en principe ils seront compatibles.

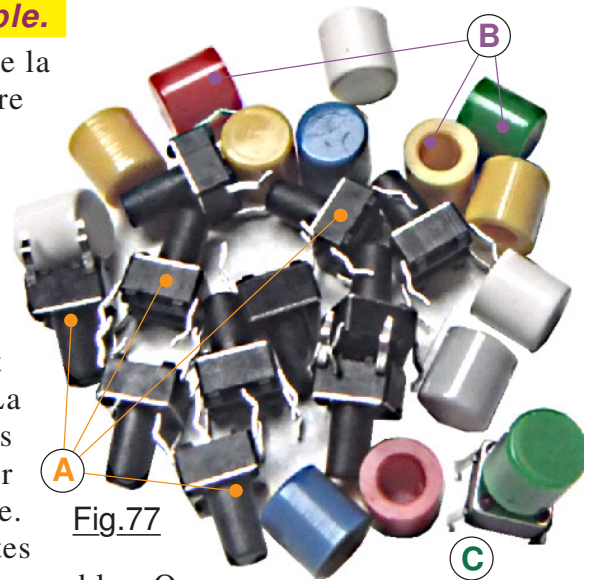


Fig.77

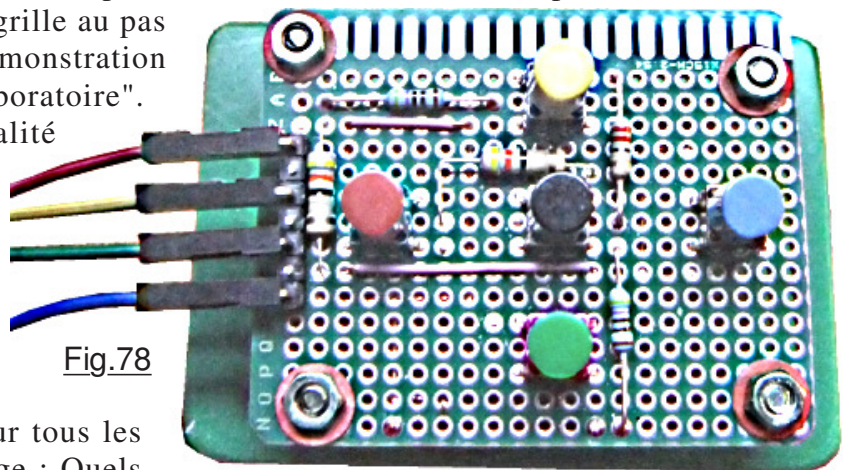


Fig.78

➤ L'écran typique durant le RUN.

Première étape du développement de la fonction [RUN](#), définir l'organisation de la page écran qui sera visualisée durant l'exécution d'un algorithme. C'est une phase cruciale car de nombreuses informations sont susceptibles d'être visualisées. Le choix de leur disposition influence directement la qualité opérationnelle du petit appareil. L'écran OLED sera également secondé par la LED triple dont le comportement informera visuellement l'opérateur du cours de déroulement de certaines étapes ou options validées. Avant de détailler les divers protocoles du mode [RUN](#), commençons par nous faire une idée de l'écran qui lui sera associé. Sur la Fig.79 on

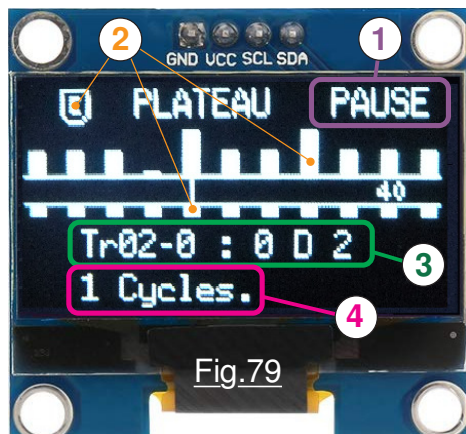


Fig.79

qui n'est ralentie que par le temps nécessaire au rafraichissement de l'écran OLED. (Environ 8 images par seconde. Ce ralentissement n'est pas fonction de la quantité d'informations présentées sur la page écran. C'est une durée propre à l'afficheur qui active ces pixels en huit balayages de sa trame à chaque rafraichissement.) Toute la zone 2 présente le carrousel avec les pions et la tête de L/E. Le mot **PLATEAU** précise qu'il s'agit de

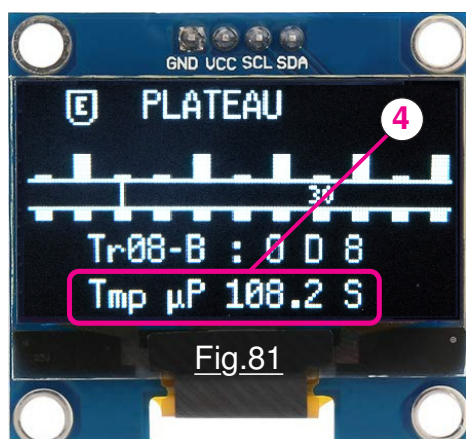


Fig.81

l'état actuel du BARILLET et non celui de son INITIALISATION préalable. Sur la ligne 3 sont affichées les instructions de l'algorithme en fonction de la TRANSITION **Tr** active et de l'état binaire du pion situé sous la tête de lecture.

➤ Comportement du clavier et du codeur rotatif.

Deuxième étape primordiale du développement de la fonction **RUN**, définir les protocoles d'utilisation de l'interface Homme/Machine. Si la machine rend compte de son état et accuse réception des consignes par l'écran et la LED triple, c'est par le

clavier et le codeur rotatif que l'opérateur donne ses ordres. Conformément aux indications du tableau Fig.82, quand le programme est en train de se dérouler et qu'il n'est pas en **PAUSE**, en tournant le codeur incrémental on fait afficher les diverses zones *en permutation circulaire* fonction du sens de rotation. **ATTENTION : Que l'on soit en mode PAUSE ou en déroulement**

Noir	4	3	4-3	2	2-4	2-3-4	Noir
------	---	---	-----	---	-----	-------	------

Fig.82

continu des instructions de l'algorithme, cliquer sur le B.P.C. fait sortir sans préavis de l'exécution et affiche le résultat du traitement. Quel que soit l'état d'affichage du tableau de la Fig.82 le

comportement du clavier sera celui résumé sur la Fig.83 avec **BP1** et **BP3** assistés par la LED tricolore. Cliquer sur **BP4** valide ou suspend le mode **PAUSE**. Appuyer sur **BP2** fait alterner dans la zone 4 le nombre de cycles effectués, ou le temps écoulé depuis le début du **RUN**. Par exemple sur la Fig.81 l'opérateur a choisi de faire afficher le temps d'exécution en temps réel. Enfoncer le **BP1** passe le déroulement de l'algorithme en fonctionnement ralenti. Dans ces conditions l'horloge cadence deux cycles par seconde. Pour avertir l'opérateur que le programme ne se déroule plus à la vitesse rapide, alors que l'écran serait noir, la LED tricolore clignote rapidement en "violet". Si on

Thèmes affichés à l'écran

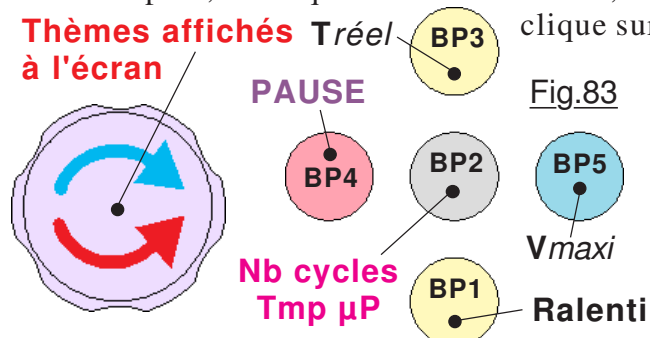


Fig.83

clique sur **BP3** le fonctionnement est ralenti à la cadence réelle de la machine électromécanique. Pour signaler ce mode la LED s'illumine en cyan. Il faut entre 3 secondes minimum et 7,4 secondes au maximum par cycle. Noter que lorsque l'un des deux ralentissements est validé, le clavier n'est pris en compte qu'à la fin du cycle. Il faut donc laisser la touche du clavier enfoncée jusqu'à ce que la LED triple s'illumine en vert pour signaler sa prise en compte. **Page 27**

Quand à **BP5**, elle impose le fonctionnement le plus rapide en annulant affichage et temporisations. Ce mode de fonctionnement est le plus rapide, car seul le traitement du microcontrôleur consomme du temps, le frein de l'affichage n'étant plus pénalisant. Pour que le programmeur puisse vérifier que l'algorithme est bien en train de se dérouler, la LED tricolore n'allume que la couleur bleue. C'est par le truchement de toutes ces teintes lumineuses résumées dans le tableau de la Fig.84 que sont différenciés les différents types d'état de l'appareil.

Couleur	Clignotement	Signification
Vert	Non	Touche clavier active.
Vert	Rapide	Attente d'une touche clavier.
Rouge	Non	B.P.C. activé.
Blanc	Non	Écran et logiciel en veille.
Violet	Rapide	Mode RALENTI.
Cyan	Non	Temps Machine Réel.
Bleu	Non	Exécution en cours. <u>Fig.84</u>

➤ La page d'ouverture du mode **RUN**.

Valider l'exécution du programme perforé dans la grille virtuelle commence par afficher la page écran de la Fig.85 qui résume l'état des diverses options mémorisées dans la machine avec en haut le rappel de la référence de l'algorithme présent. *(Celui qui devait fonctionner du premier coup tellement l'idée était simple et qui s'engueule de s'engueule nous ruine notre patience et bousille les chakras !)* Nous savons que dans le mode PAS à PAS seul **BP4** inversera ce mode, les quatre autres touches déclenchant un cycle de plus ... jusqu'à éventuellement rencontrer le "**F**" qui fait sortir du programme. Les deux options de temporisation **Tréel** et **Ralenti** s'excluent mutuellement, et le **BP5** les annule, suspend le mode PAS à PAS et éteint l'écran. Si une **BORNE** de sortie prématurée a été initialisée, comme on le constate sur la Fig.85 sa valeur n'est pas indiquée, car elle peut être très importante et son affichage déborderait de l'écran. On se contente dans ce cas d'afficher **OUI**. On peut également

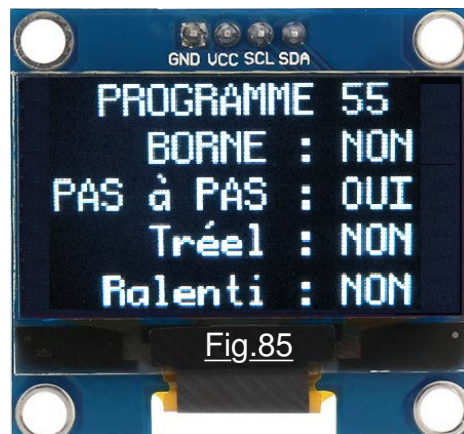


Fig.85

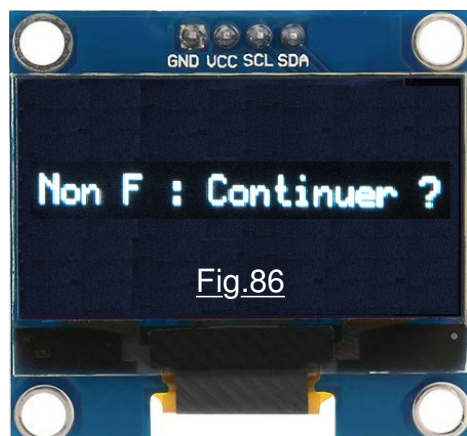


Fig.86

avoir positionné l'option **SURVEILLE** qui exclue le mode PAS à PAS. Si la valeur de cette option est supérieur à zéro, l'affichage de la ligne **PAS à PAS : NON** est remplacé par **Surveillance : OUI**. La LED tricolore clignote en vert et n'importe quelle touche du petit clavier activera le déroulement de l'algorithme. Si l'option "Ignorer PAS de FIN dans l'algorithme" est validée et que "**F**" n'est présent dans aucune des lignes du programme, l'alerte de la Fig.86 s'affiche sur l'écran. Toute touche autre que le **BP3** sera considérée comme une réponse négative et engendrera le retour au **MENU de BASE**. L'acceptation avec **BP3** enchaîne alors sur la page de la Fig.85 et attente d'une action sur le clavier.

➤ Les diverses façon de sortir du mode **RUN**.

Victorieuses ou pas vraiment comme prévues. L'expérience montre que c'est la deuxième qui est statistiquement la plus probable. Trois scénarios de fins prévues sont potentiels. Le premier consiste à cliquer sur le B.P.C. car nous savons que "**F**" n'est pas présent dans l'algorithme, ou que par nature, même à cadence maximale il va falloir beaucoup de temps et que l'on désire reprendre la main. Pour le deuxième cas, le "**F**" a été rencontré et engendre la sortie. *(Et là nous attend la surprise pour l'état du plateau qui s'acharne à montrer des pions "pas comme il faut".)* Enfin, une **BORNE** est programmée et engendre la sortie au nombre de cycles d'horloge initialisé. Dans ces trois cas il y a affichage du résultat du traitement sous la forme présentée en Fig.87 ou en Fig.88 si on clique sur **BP1**. On retrouve le comportement

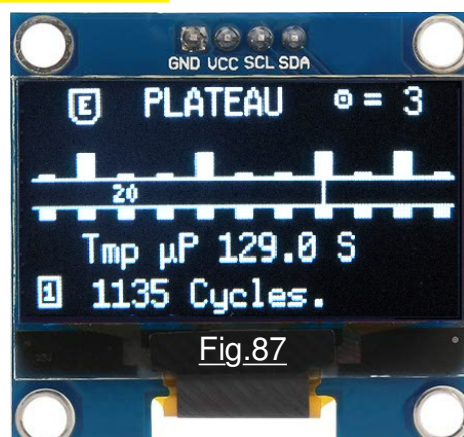


Fig.87

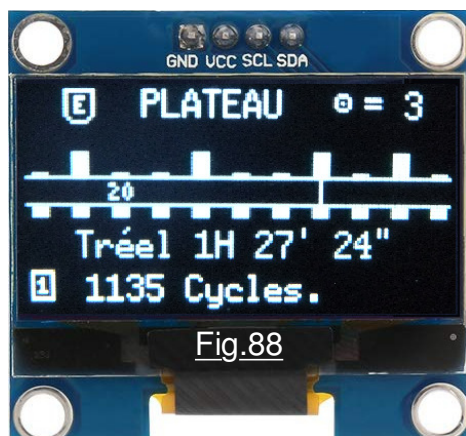


Fig.88

le RUN. Lorsque le programme arrive au jalon initialisé il se met en PAUSE. On clique alors sur BP4 pour valider le mode PAS à PAS et ainsi analyser sereinement le déroulement de la tragédie. On dispose aussi du mode SURVEILLE comme outil de diagnostic sans compter ... que l'on peut aussi vérifier le bienfondé de la feuille perforée virtuelle.

du petit clavier indiqué dans le chapitre *La fonction PLATEAU* donné en page 22. La sortie la plus tragique est celle de la Fig.89 qui ne fait qu'afficher un écran d'alerte. La LED tricolore clignote en vert et annonce le retour direct à la case départ, c'est à dire au MENU de BASE. Si un tel cas se produit, comme nous savons à quel moment cette calamité va se produire, on peut facilement anticiper. On commence par placer une BORNE un peu avant l'incident. Puis on déclenche

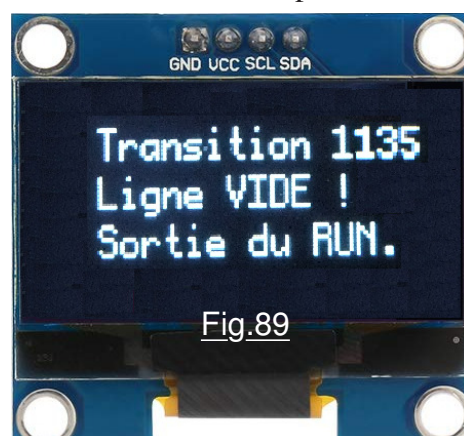


Fig.89

20 x 3 x 2 = 66 !

C'est assez nouveau, ou plus exactement je viens juste de m'en rendre compte, et les conséquences de cette mathématique étrange ne sont pas dérisoires. Bon, un petit retour en arrière s'impose pour déterminer l'*origine de cette grossière erreur*. C'est durant la rédaction du démonstrateur P08 que c'est produite l'erreur fatale. À ce stade optimiser l'utilisation de l'espace disponible en EEPROM tournait à l'obsession. La répartition des zones a été faite avec soin, laissant le début des cellules pour y loger un maximum de textes. C'est ainsi que la zone bleue pour les algorithmes a été réservée en premier, puis la zone verte pour y loger les 52 octets représentant le BARILLET. Puis, contre la zone réservée au carrousel a été placée la zone jaune réservée au programme ÉTENDU. Enfin, tout ce qui restait comme place dans la



Fig.90

zone marron a été saturée par du texte. TOTALEMENT utilisée l'EEPROM ! Dans cette phase du développement, j'avais en tête la taille de 66 octets pour un programme "standard", c'est à ce stade que j'ai commis ce stupide égarement, car un algorithme ÉTENDU implique 120 octets et non 66. Que faire ? Réparer cette bêtise consisterait à diminuer la taille des textes de la zone marron et en enlever 54. Je n'ai pas osé faire ce choix, car il aurait alors fallu reprendre tous les affichages de P08 jusqu'à P13 actuel pour ne pas qu'il ne se produise des incohérences. Un tel travail ne me fait pas reculer du tout. En revanche, quand j'ai constaté à quel point l'optimisation des textes pour gagner trois ou quatre emplacements en EEPROM a engendré des "effets boule de neige", pour 54 caractères je n'ai pas osé prendre un tel risque. *La solution retenue consiste à récupérer en EEPROM la zone verte du BARILLET pour y loger la fin du programme ÉTENDU*. Du coup, dans ce mode il devient interdit (*Donc impossible par programme.*) de SAUVEgarder ou de RECHARGER un BARILLET, tentative qui engendrera par programme l'alerte de la Fig.90 pour ces deux tentatives. Enfin, comme la sauvegarde d'un programme ÉTENDU écrase un éventuel BARILLET préservé en EEPROM, l'écran de la Fig.44 en page 15 est complété par le texte "(Perte du BARILLET.)".

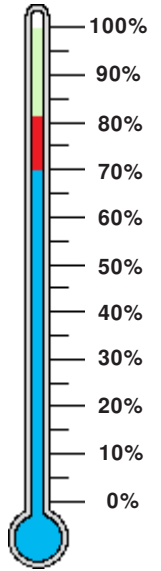


Ben Môa môa j'ai toujours pensé que les calculatruc et les programmastuces ne vont pas du tout ensemble !

09) Ajout de la gestion des **OPTIONS** dans le logiciel.

Fig.91

Comme le montre le thermomètre logiciel de la Fig.91 la zone rouge qui fait 12% est celle engloutie par la fonction **RUN**. Si l'on revient sur mon estimation qui accompagne la Fig.76 de la page 26, on constate que j'étais un peu optimiste, mais l'ordre de grandeur annoncé n'était pas à franchement parler complètement erroné. La zone verte qui reste pour pouvoir loger les routines traitant les **OPTIONS** semble bien faible. Et bien rassurez-vous, je suis persuadé que cet espace encore disponible sera largement suffisant. En effet, faire fonctionner l'exécution imposait de très nombreuses procédures complexes et plusieurs écrans très consommateurs d'octets. Naturellement pour les **OPTIONS** il va y avoir quelques nouvelles pages affichées. En revanche, les procédures vont à mon sens rester simples et ne comporteront pas de grands nombres d'instructions. Pour ma part le moral est au beau fixe. J'y crois en j'envisage la possibilité de faire une page écran d'accueil sympa avec indication de la version du logiciel. Passons aux actes, sans plus tarder car les items à mettre en place sont assez nombreux et l'arbre combinatoire des fonctions à émuler présente vraiment beaucoup de branches sur le papier.



➤ Les options de base.

Celles qui sont le plus utilisées sont donc celles placées au début du menu des **OPTIONS** dont la Fig.92 présente l'allure. C'est le démonstrateur **P14_Introduction_des_OPTIONS.ino** qui se charge d'en assurer le développement et la validation. Quand on valide l'item **PAUSES** l'écran s'efface et affiche en son centre le texte **Pas à PAS : OUI**. Si on tourne le capteur rotatif, les options **OUI** et **NON** vont alterner à l'écran. En cliquant sur le B.P.C. on entérine la valeur initialisée et l'écran affiche l'information **Retour MENU de BASE**. La LED triple s'illumine en jaune non clignotant. Cette possibilité qui n'est pas mentionnée dans le tableau de la Fig.84 de la page 29 signifie que l'on se trouve dans l'affichage du résumé des options. Si on clique sur le B.P.C. il y aura retour au **MENU de BASE**. Au contraire, si l'on tourne le capteur rotatif on va faire défiler les quatre autres

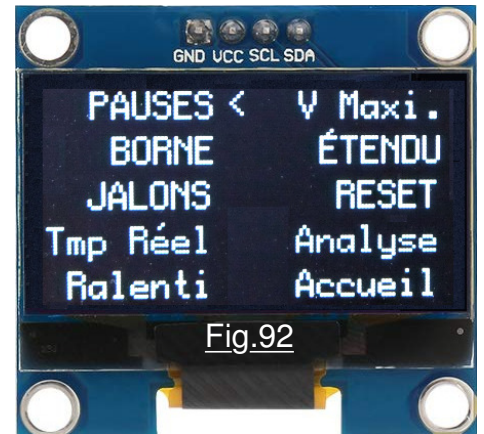


Fig.92

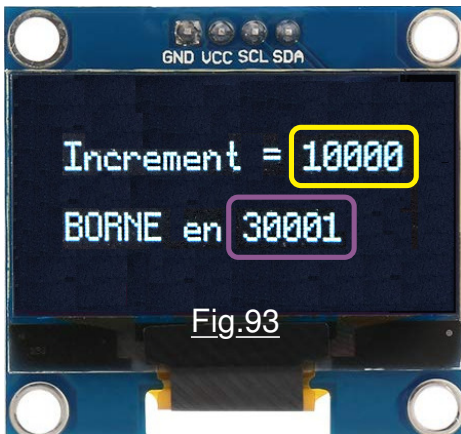


Fig.93

pages d'écran qui résument l'état actuel des initialisations des options. La première présente l'état des trois premiers items du menu des **OPTIONS**. Si l'on clique sur le B.P.C. alors que l'une des quatre pages est affichée "avec la LED jaune" il y a retour dans le menu des **OPTIONS** et l'on peut alors en conditionner une autre. Par exemple on valide l'item **BORNE** provoquant l'affichage de l'écran Fig.93 avec par défaut deux valeurs unitaires. On doit pouvoir sélectionner des valeurs faibles autant que des valeurs très élevées. Aussi, en cliquant sur l'un des B.P. du clavier on impose la valeur des **Incréments** de modification de la valeur de **BORNE** quand on fait tourner le codeur rotatif. Noter qu'avec le bouton poussoir

BP5 on peut sélectionner trois valeurs en permutations circulaires. Pour que l'opérateur puisse savoir à tout moment la valeur actuelle de l'**Incrément** l'écran la présente au dessus de **BORNE**. Comme pour tous les Items on valide avec le B.P.C. sachant que par Décrément on peut obtenir

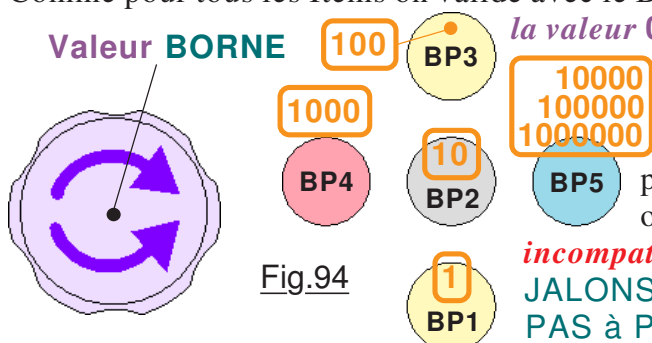


Fig.94

la valeur 0 qui invalide une **BORNE** éventuellement initialisée précédemment. Nous pouvons vérifier sur la Fig.95 **A** qu'initialiser une **BORNE** n'efface pas pour autant le mode **PAS à PAS** s'il est validé pas plus que préciser une valeur pour **JALONS**. Sur la Fig.95 **B** on observe par contre que **JALONS et BORNE sont incompatibles et s'excluent mutuellement**. Pour mémoire, avec **JALONS** le programme va fonctionner comme avec le mode **PAS à PAS** mais les **PAUSES** ne seront

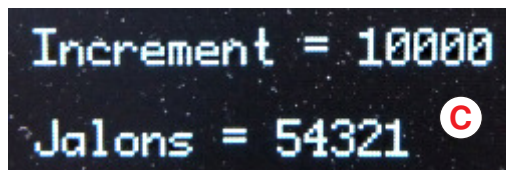
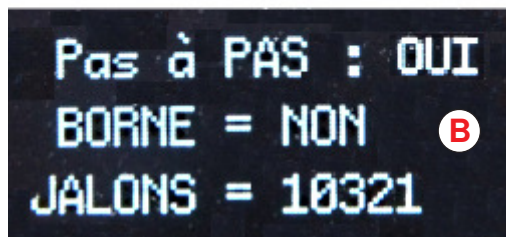
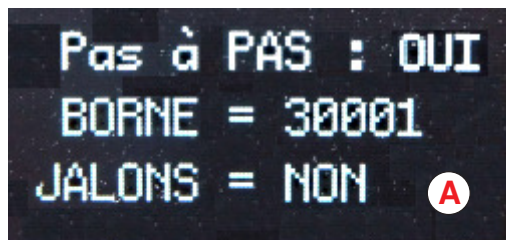


Fig.95

effectives que tous les **JALONS** cycles. On réalise de grands sauts dont l'ampleur peut être considérable à convenance. Noter que le **BP4** se comporte comme une bascule pour **PAUSE** mais n'affecte pas la valeur de **JALONS** car pour suspendre ces grands sauts il faut annuler la valeur qui serait perdue. Le protocole pour saisir la valeur des **JALONS** est strictement identique à celui de l'initialisation de **BORNE**, les informations de la Fig.94 restant totalement applicables. La Fig.95 **C** présentant l'écran de saisie.

➤ Gestion du cadencement de l'HORLOGE.

Dévolu aux trois Items qui suivent dans la liste de la Fig.92 on trouve les trois frères ennemis, qui s'excluent mutuellement. Ces trois options servent à gérer la rapidité du déroulement de l'algorithme en mode **RUN**. Nous comprenons assez facilement qu'avec l'item **Tmp Réel** l'horloge virtuelle va fonctionner à exactement la cadence de l'horloge matérielle de la machine électromécanique. Comme nous le savons c'est la vitesse de traitement des algorithmes la plus lente. Si l'on vote pour **Ralenti**, la Fig.96 montre que **Tmp Réel** est alors désactivée. Dans ces conditions la LED triple ne sera plus éclairée en cyan continu mais en violet scintillant. La fréquence horloge sera de 0,5Hz avec deux cycles réalisés par seconde. Enfin, l'option **V Maxi.** du menu des **OPTIONS** annule tous les ralentissements ainsi que le mode **PAS à PAS** s'il était initialisé. C'est bien évidemment le mode le plus rapide pour dérouler un algorithme. La machine virtuelle n'est alors significativement ralentie que par l'affichage éventuel des informations sur l'écran OLED si ce dernier n'est pas Noir, option que l'on obtient nous le savons en tournant le codeur rotatif.

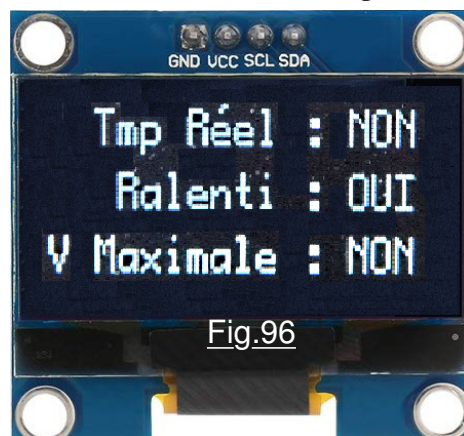


Fig.96

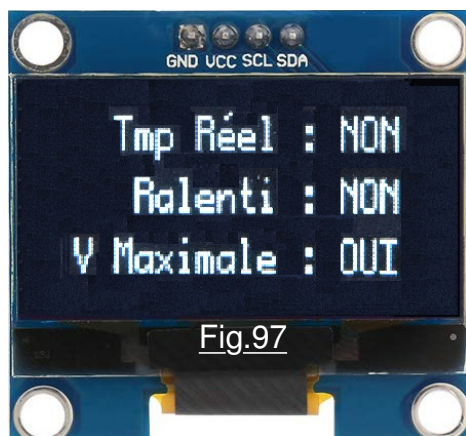


Fig.97

➤ Nouvelle page-écran en mode **RUN**.

Développer les fonctions et les protocoles des nombreuses **OPTIONS** a clairement fait ressortir un manque dans la visualisation du plateau de la machine durant le **RUN**. En effet, il est rapidement apparu que pouvoir remplacer la visualisation du carrousel par la vue d'ensemble du **BARILLET** de la machine serait un plus incontestable. Aussi, dans cette nouvelle version de la machine virtuelle, quand on tourne le codeur rotatif durant un **RUN** on termine les permutations circulaires des écrans par celui de la Fig.98 qui présente l'ensemble du plateau avec la position de l'origine et surtout celle de la Tête de L/E. En soit ce n'est pas une révolution. Toutefois, dans certains cas comme pour celui de l'emplacement n°0 ce type de visuel s'avère très pertinent. Et comme je supposais qu'il y aurait assez de place pour "tout émuler" ... au diable l'avarice. Dans ce type de représentation le plateau de la machine est immobile et ce sont les deux curseurs qui indexent la position de la tête de L/E qui se déplacent à la vitesse du cadencement de la mécanique virtuelle. Il est évident que sur cette grille les deux curseurs qui indiquent les coordonnées de l'Origine arbitraire sont immobiles.

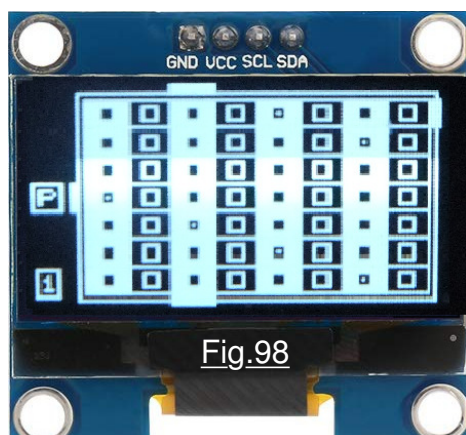


Fig.98

➤ Option Machine étendue.

Valider cette possibilité engendre immédiatement deux effets. Le premier sera de modifier immédiatement la référence du programme avec la valeur 255 si un algorithme était déjà présent en mémoire dédiée. Si tel est le cas ce programme ne sera pas effacé, car il est tout à fait envisageable d'utiliser un algorithme de taille standard, et de passer en mode PGM ÉTENDU pour le perfectionner. Le deuxième effet réside dans l'interdiction en machine étendue de sauvegarder ou de recharger un BARILLET.

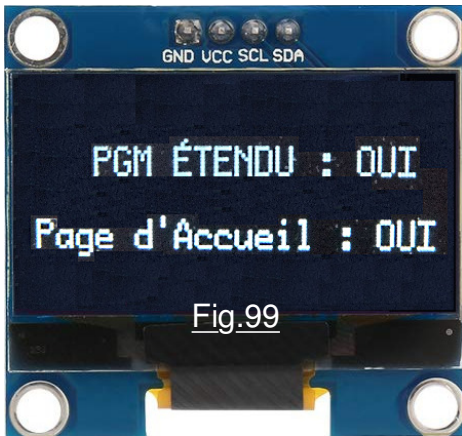


Fig.99

Dans les deux cas une tentative se soldera par le message d'erreur **Pas en PGM ÉTENDU !**. (Voir la Fig.90 dans l'encadré explicatif de la page 30.) Quitter cette page-écran d'erreur se fait avec l'une quelconque des touches du clavier comme invite à le faire la LED tricolore qui clignote assez rapidement en vert.

➤ Option Accueil.

Persuadé que pouvoir démarrer cette petite machine de Turing avec un écran d'accueil spécifique serait possible, l'idée consiste à minima à présenter un écran qui précise la référence du logiciel "tournant" dans le microcontrôleur. Ainsi, si par la suite le programme était révisé, l'utilisateur pourrait facilement vérifier s'il possède bien la dernière version. Je stipule "à minima" car j'envisage sérieusement d'optimiser à outrance le code pour que cette page d'Accueil soit au final plus "graphique". Toutefois, avec ce démonstrateur on va se contenter de ce que montre la Fig.100 qui sera déjà pas mal si l'on ne peut faire mieux. Dans les diverses pages de résumé de l'état des options, on peut se demander pourquoi l'affichage de l'accueil voisine avec celui de l'option de PGM ÉTENDU qui manifestement n'a aucun rapport. C'est tout simplement qu'il faut présenter l'état de toutes les OPTIONS sur un minimum de pages. Aussi, pour leur répartition, si une certaine logique s'impose, elle doit toutefois coexister avec des compromis.

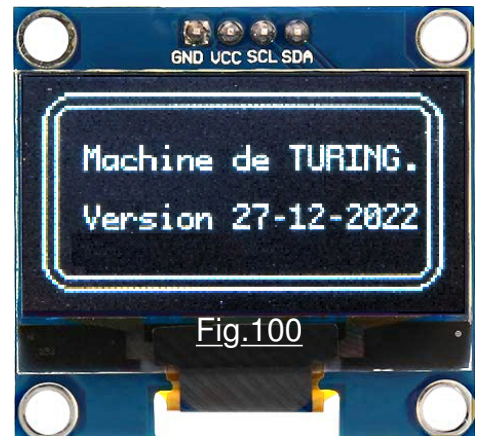


Fig.100

ATTENTION : Avec le démonstrateur de développement des options **P14_Introduction_des_OPTIONS.ino** lorsque vous allez tester la possibilité d'afficher la page d'Accueil lors du RESET, vous constaterez que l'écran est un peu plus "bavard" que celui de la Fig.100 qui dans la pratique correspond à l'écran sur le programme Arduino définitif. Comme il a fallu gagner un peu de place, le texte "Version du" a été supprimé. Donc ne tenez pas compte de la divergence entre la réalité et cette copie d'écran effectuée pour le didacticiel.

➤ Les options de chargement automatique sur RESET.

Pouvoir recharger automatiquement un algorithme et un BARILLET associé sur RESET demeure un critère de convivialité important. Du reste on doit avoir le choix libre entre l'une seule de ces deux entités si on le désire. Par exemple sur la Fig.101 qui montre la page-écran qui s'ouvre

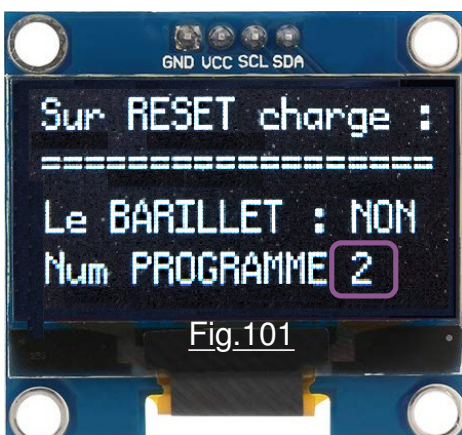


Fig.101

avec RESET on ne désire recharger que l'emplacement n°2 mais pas Le BARILLET. Pour saisir ces deux options le protocole est un peu particulier. En tournant le bouton du codeur rotatif on modifie la valeur de l'emplacement EEPROM qui sera chargé sur un RESET.

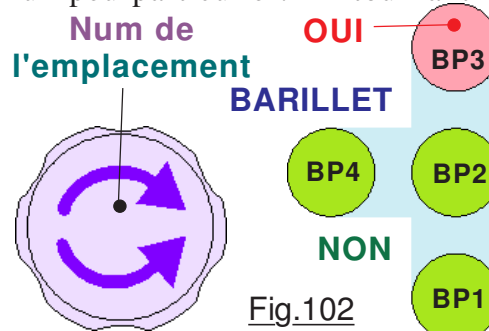


Fig.102

C'est avec l'une des cinq touches du petit clavier que l'on précise si l'on désire charger Le BARILLET. BP3 impose OUI alors que

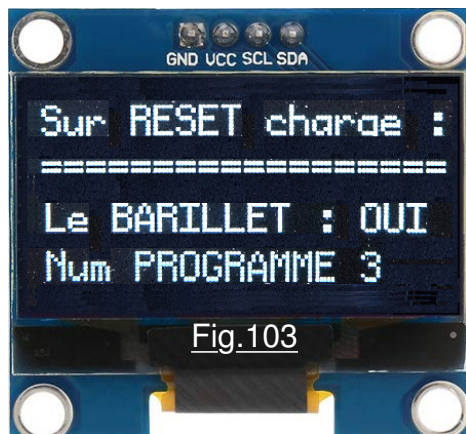


Fig.103

RESET, dans l'écran dédié de l'état des OPTIONS on obtient une page du type de celle de la Fig.103 montrant le cas du choix d'un algorithme standard avec ici le rechargement de la configuration du BARILLET. Si on opte pour le programme ÉTENDU il y a désactivation automatique du chargement d'un BARILLET si ce dernier était validé précédemment. Enfin, si sur un RESET il y a chargement automatique, l'opérateur en est averti par un écran tel que celui de la Fig.104 qui présente le cas du transfert d'un programme ÉTENDU qui fait automatiquement passer la machine dans ce mode avec affichage comme référence de l'algorithme la valeur particulière de 255.

les autres boutons **BP1**, **BP2**, **BP4** et **BP5** annulent cette initialisation. En tournant le codeur rotatif on fait apparaître dans l'ordre **///**, **0** à **9**, **20Tr** et **NON**. Le premier symbole par défaut est **///** correspond à la fuite, c'est à dire une sortie sans modifier l'option. **NON** précise on s'en doute qu'il ne faut pas charger d'algorithme sur un RESET. C'est la valeur à valider pour annuler un chargement actuellement validé. La valeur **20Tr** précise que l'on désire charger un programme ÉTENDU comme le souligne de texte (**PGM ETENDU.**) qui accompagne le texte **20Tr**. Lorsque l'on a validé des options de rechargement automatique sur

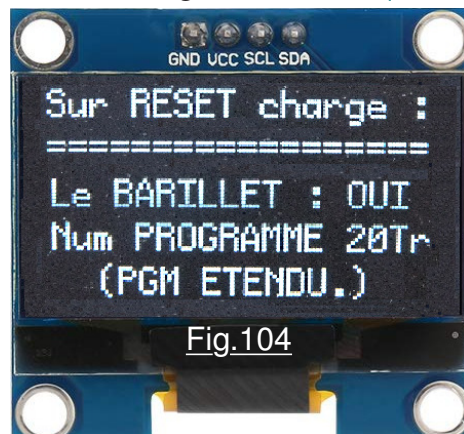


Fig.104

ATTENTION : Si un programme cohérent n'a pas été enregistré en mode **ÉTENDU** on peut fort bien recharger des codes correspondant à un **BARILLET** avec appel à des transitions du genre 255 par exemple.

➤ L'option ANALYSE.

Accompagnée d'une liste d'items relativement courte puisqu'il n'y en a que cinq, c'est pourtant une facette très gloutonne du logiciel en espace réservé au programme car certaines de ces fonctions imposent plusieurs écrans boulimiques en octets. L'effet du premier Item de la liste est élémentaire. Quand on clique sur le l'un quelconque des **BP** du petit clavier alors que l'index est face à **Verif NON F : OUI** on inverse son option. En tournant le bouton du codeur rotatif on indexe l'une des autres lignes dans la liste de cette page-écran. Ce comportement est valable pour les trois Items :

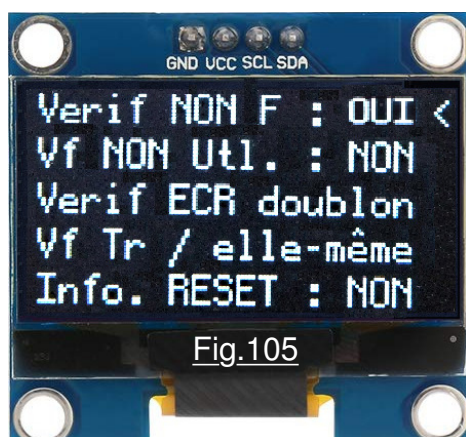


Fig.105

- **Verif NON F :**
 - **Vf NON Utl. :**
 - **Info.RESET :**
- Si on a validé

l'option **Verif NON F** chaque fois que l'on activera **RUN** il y aura vérification de la présence au moins d'un "**F**" dans le programme. Si ce n'est pas le cas le logiciel affichera **NON F : Continuer ?** puis il attendra une consigne de l'opérateur. Une confirmation engagera le fonctionnement

automatique alors que **NON** ramènera au **MENU de BASE**. L'Item **Vf NON Utl.** est une contraction de **Vérifier les ligne NON Utilisées**. Quand cette possibilité est validée, La sortie du programme, même si elle est anticipée par l'opérateur, est suivie d'un écran du type de celui de la Fig.106 qui n'indiquera que la première ligne de l'algorithme du programme qui sera détectée comme non vide et NON utilisée. Ce n'est qu'une information qui ne sera pertinente que dans des cas particuliers de mise au point d'un algorithme réticent.



Fig.106

L'option **Info.RESET** est directement liée au rechargement d'un algorithme ou d'un BARILLET lors du RESET. L'écran d'information de la Fig.104 généré lors d'un RESET impose au programmeur d'accuser réception en cliquant sur l'une des touches du clavier. C'est utile lorsque l'on a validé l'une de ces possibilités et que l'on reprend la machine suite à plusieurs semaines de non utilisation. En revanche, lors de la mise au point d'algorithme, cette page écran peut rapidement devenir indigeste. Affecter **OUI** ou **NON** à cet Item nous rend totalement libre de voter pour le meilleur candidat.

➤ La vérification des doublons dans le programme.

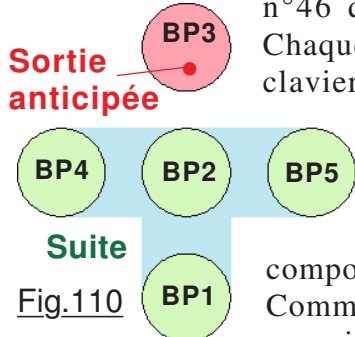
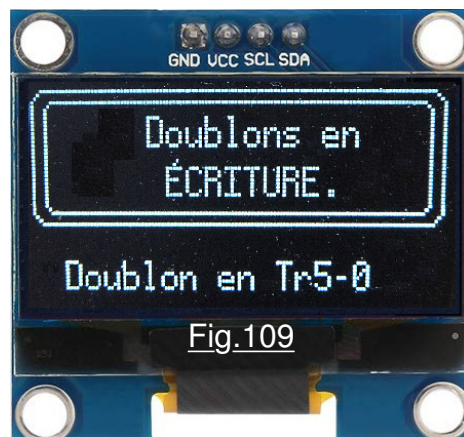
Globalement, mis à part des programmes tel que l'algorithme n°46 qui avait pour but de percer des "sinusoïdes" sur la feuille de programme, ÉCRIRE des état inutiles n'est absolument pas pertinent car il impose de perforer des emplacements pour rien dans la grille. Par exemple Imposer d'écrire un "0" sur la ligne de LECTURE d'un "0" pour la transition concernée. En général un tel



"doublon" correspond à une étourderie soit dans la conception de l'algorithme, soit lors de sa saisie dans **MODIFIER** ou **RÉDIGER**. Quand on valide la fonction **Verif ECR doublon** avec le B.P.C. en fonction de ce qui se trouve en mémoire dédiée

on peut avoir l'affichage de la Fig.107 qui correspond au cas préféré de l'opérateur et qui n'appelle pas de commentaire. Si une écriture inutile est présente dans l'algorithme elle sera affichée comme sur la Fig.108 pour prévenir le

programmeur qui devra revoir sa copie. On peut fort bien se retrouver avec un nombre important "de boulettes", par exemple on a chargé en mémoire le programme utilisateur de référence n°46 qui comporte dix écritures inutiles. Chaque fois que l'on accuse réception sur le clavier l'analyse affiche le doublon suivant.

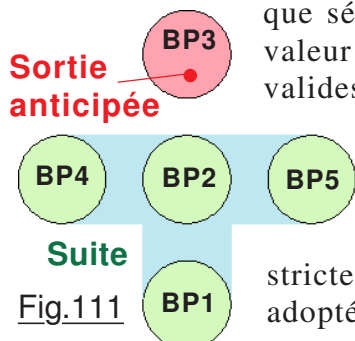


Aussi, pour pouvoir se sauver de ce type de situation le **BP3** engendre la sortie anticipée de l'analyse. La Fig.110 proposée ci-contre résume le comportement du petit clavier à cinq touches. Comme pour toutes les autres options on peut revenir au **MENU de BASE** ou à la liste des

OPTIONS. Noter que si l'on est dans le cas de la Fig.109 et que l'on se fait lister la totalité des écritures inutiles, on sort alors par l'écran de la Fig.107 dont il ne faudra pas tenir compte.

➤ La vérification des transitions suspectes dans le programme.

Autant "perforer" des trous inutiles sera sans conséquence sur le déroulement du programme, autant introduire des transitions incorrectes affectera le fonctionnement de l'algorithme. Alors avoir la possibilité d'analyser le programme pour faire émerger ce type d'erreur peut s'avérer plus



que séduisant. Quand on rédige l'algorithme, il est impossible d'indiquer une valeur de transition erronée car le codeur rotatif ne présente que les options valides. Par contre, brancher une transition sur elle-même peut correspondre à une erreur de conception ou une erreur de saisie. Ce n'est pas forcément involontaire. Il sera toutefois parfois utile de soumettre l'algorithme à une telle recherche, au programmeur de déduire si c'est normal ou involontaire. Le comportement du clavier reproduit en Fig.111 est strictement identique, le protocole pour cette fonction étant le même que celui adopté pour les écritures inutiles. Sur la Fig.112 nous avons le



Fig.112

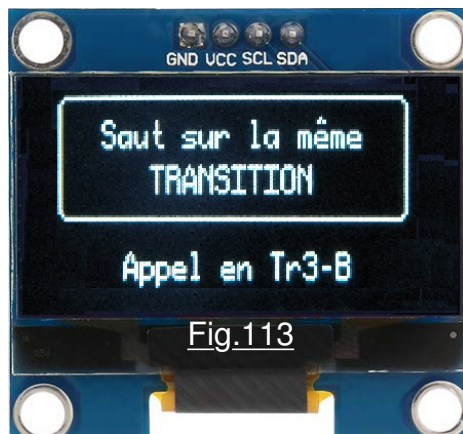


Fig.113

cas d'un programme sans problème. La Fig.113 quand à elle est représentative d'un algorithme avec une ou des transitions suspectes. C'est alors au programmeur de vérifier le bienfondé de cette alerte. D'une façon générale transiter sur la même Tr sera une étourderie à annuler ...

➤ Quelques améliorations apportées au démonstrateur P13.

Visiblement il restait assez de place pour ajouter quelques petites améliorations au démonstrateur précédent. Certaines sont "dérisoires", d'autres plus consistantes. Par exemple sur la Fig.114

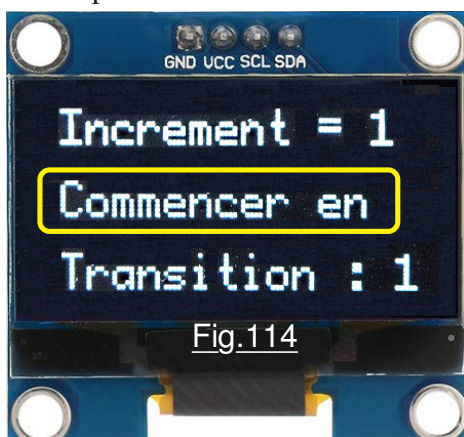


Fig.114

on peut voir dans l'encadré jaune qu'une ligne de texte a été ajoutée dans la page qui s'ouvre avec **MODIFIER** du menu **PROGRAMME**. Ce n'est pas une révolution, mais le texte d'ensemble est plus précis. Maintenant, quand dans le **MENU de BASE** on clique sur **BP4** on obtient l'écran de la Fig.115 qui détermine la place de mémoire dynamique qui reste entre la **PILE** et le **TAS**. Cette information ne concerne que le programmeur

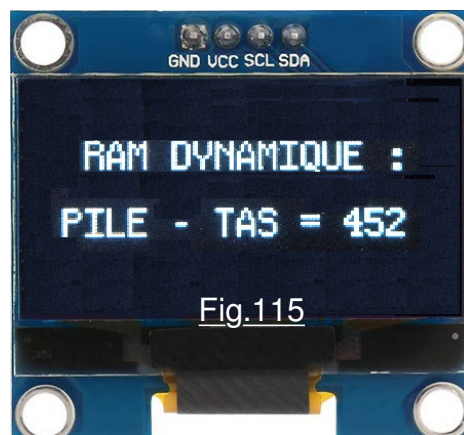


Fig.115

et a été abordée dans les pages 37 et 38 du tutoriel sur la machine élémentaire. Enfin, dans les petits plus on terminera par l'option

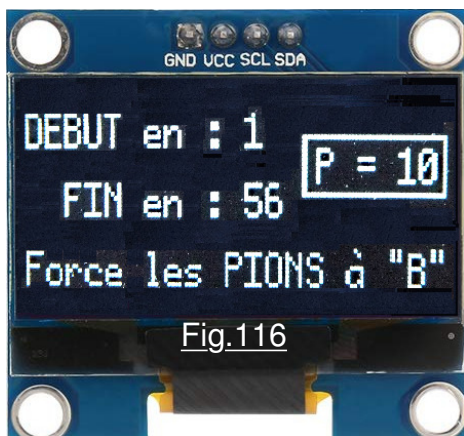


Fig.116

Sature de remplissage du carrousel par des états identiques. Cette option concerne le menu **BARILLET**. Actuellement, ce que montre la Fig.116, on peut remplir une zone inférieure à la taille du plateau dont on précise le **DEBUT** et la **FIN**. Le codeur rotatif permet de choisir entre "**B**", "**0**", "**1**" ou la **Fuite** si on a indexé le changement de valeur des pions. Il incrémente ou décrémente la valeur de **DEBUT** ou de **FIN** si ces Items sont indexés. La touche **BP4** ou la touche **BP5** permettent de sélectionner un **Pas de 1** ou de **10**. Les trois touches verticales indexent respectivement l'état des **PIONS** pour **BP1**, La valeur de la **FIN** pour **BP2** et celle de **DEBUT** pour **BP3**. La Fig.118 montre le résultat pour le choix de la Fig.117.

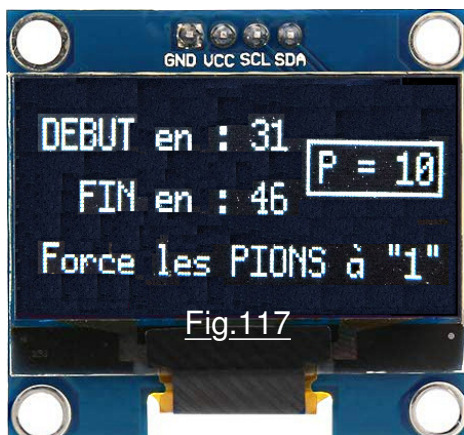


Fig.117

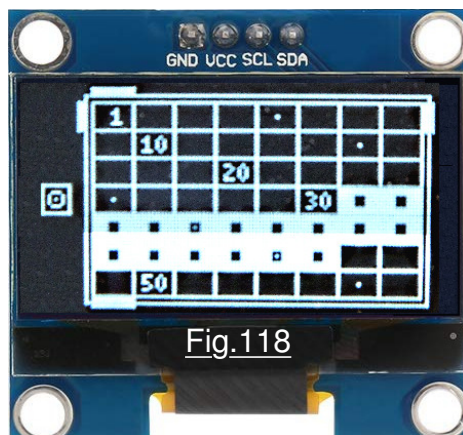
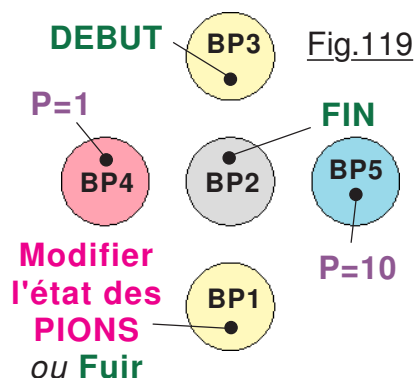
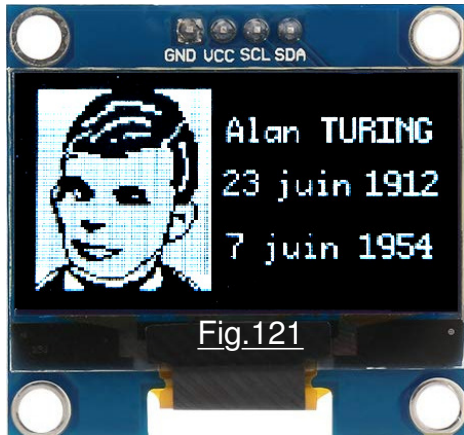


Fig.118

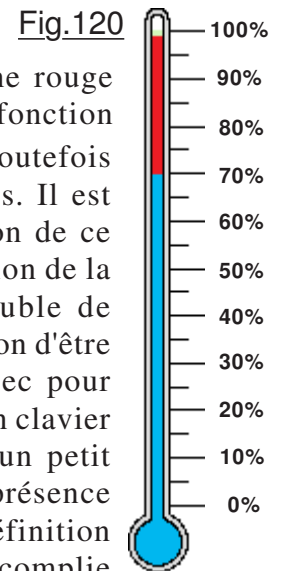


10) Un petit délire de luxe.

Manifestement, sur le thermomètre de la Fig.120 on observe que la zone rouge titille allègrement les 28%, c'est est celle goinférée sans vergogne par la fonction de gestion des **OPTIONS**. Une boulimie presque scandaleuse qui se justifie toutefois par le nombre d'écrans à émuler et des traitements pas forcément élémentaires. Il est



assez singulier de constater que cette gestion de ce qui n'est au fond que des "facilités" d'utilisation de la machine consomme largement plus du double de l'implémentation du **RUN** qui concerne la raison d'être de cet appareil. C'est un peu inhabituel, avec pour justification la qualité opérationnelle. Avec un clavier nain, un bouton rotatif, une LED triple et un petit écran, il fallait faire aussi bien qu'avec la présence d'un clavier étendu et d'un écran à haute définition sur la machine élémentaire. La mission est accomplie et la version actuelle du démonstrateur valide une technique très agréable à utiliser. Il reste encore un petit 2% d'espace que je me



suis réservé depuis longtemps avec l'espoir de pouvoir intégrer un hommage graphique à Alan TURING qui mériterait plus que fortement un prix Nobel, fut-il à titre posthume, vu l'apport formidable au genre humain dont il a fait preuve, et pour lequel il a été ignoré, et méprisé comme un pariât pour son homosexualité. Pas un homme à mon sens a autant contribué à l'avenir des habitants de cette planète.

➤ **Un hommage qui s'impose à la mémoire d'Alan Turing.**

L'image de la Fig.121 n'est que le pâle reflet de l'admiration sans limite que je vous au génie de ce scientifique qui a contribué à faire évoluer de façon considérables les sciences et les techniques de son époque, et qui contribueront encore de nombreuses années à influencer l'avenir de notre Terre. Ce portrait d'une définition dérisoire d'à peine 3584 Pixels codés sur 448 octets permettra malgré tout d'en graver son "histoire" dans du silicium dont l'architecture est son enfant. Pour construire ce petit dessin, le démonstrateur **P15_Page_Turing.ino** a servi à écrire et peaufiner la table des valeurs qui constituent le portrait en noir et blanc. Deux stratégies pour générer cette image étaient possibles. La première consiste à créer un tableau d'octets qui par nature sera logé dans la mémoire dynamique. Hors, ce que montre la Fig.115, le logiciel de **P14** ne laisse entre la PILE et le TAS que 452 octets pour loger les adresses de retour de sous-routines et les variables locales. C'est largement suffisant pour assurer la stabilité du programme. Si dans ces 452 octets on en enlève 448 pour loger l'image, il n'en reste plus que quatre ! Bref, cette approche est totalement inenvisageable. Deuxième solution : Imposer à cette table de valeurs d'être incluse dans la mémoire réservée au programme. Cette SDRAM non volatile n'est pas de la RAM et comme c'est le cas pour l'EEPROM ne doit pas être écrite des millions de fois, sa durée de vie étant estimée à 100000 écriture garanties. Il ne faut pas passer son temps à en modifier sans arrêt les valeurs, comme on le fait dans la RAM prévue à cet effet. Il se trouve que notre image est statique, taillée dans le marbre. Aussi, quand après une galère pour coder les 448 octets elle est construite, on n'y touchera plus. (*Sauf chaque fois que le programme sera recompilé suite à une modification de son code source.*) Inclure la table des PIXELs dans le logiciel consomme 452 octets, et encore, la largeur a été limitée à 56 points. Ce dessin a été largement réduit par rapport à celui du PAPILLON. Puis l'instruction **u8g.drawXBMP(0, 0, 56, 64, DessinBitMap)**; qui écrit les points lumineux dans la matrice de l'afficheur a été ajoutée au programme. Cette instruction prend 340 octets. Ajoutez les trois petits textes avec les instructions de positionnement et les maigres 2% qui restaient sont largement dépassés. Pour optimiser les encombrements, j'ai tenté de diviser l'image en deux.

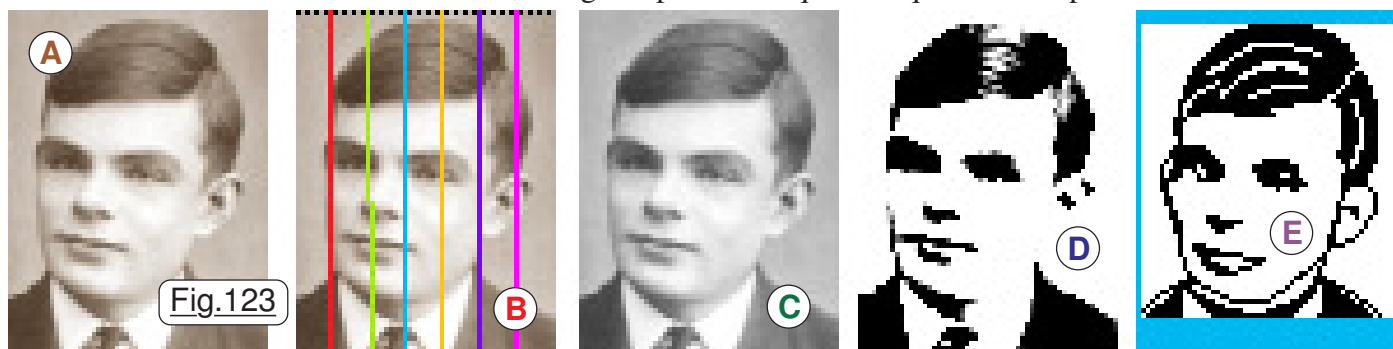


La moitié logée dans le programme et le reste en mémoire dynamique. Peine perdue, car la place gagnée pour le programme est plus que consommée par la deuxième instruction qui elle aussi consomme ses 300 octets. Bref, place insuffisante veut dire : *Soit tu oublie, soit tu fais de la place !*

Après diverses optimisations d'optimisations sur les optimisations, quelques octets ont été grappillés ici et là. La procédure qui affichait l'écran de la Fig.115 a été enlevée et une ou deux pages sur l'afficheur OLED ont été un peu simplifiées. Enfin le compilateur a accepté de traiter le programme ultime `P16_Machine_de_TURING.ino` sans afficher l'effroyable texte "Fichier trop gros".

➤ Le pinceau informatique.

Avec une image de 56 x 64 points on est bien loin de la Joconde, et la misère de la Fig.121 est très loin de représenter celle de la Fig.122 couleur sépia d'une autre époque. Pourtant, pour arriver à ce tristounet résultat la route à emprunter est assez indigeste. Pour compléter les explications données dans le petit livret [Bibliothèque U8glib.pdf](#) on va décortiquer sur la Fig.123 les étapes qui conduisent au résultat final. La première phase consiste à créer une *image Noir et Blanc qui doit impérativement contenir un nombre entier d'octets en largeur, soit un nombre de points multiple de huit*. Pour la hauteur le nombre de lignes peut être quelconque. Ici 64 pour couvrir la hauteur



complète de l'afficheur utilisé. La première étape en **A** consiste à redimensionner l'image pour lui affecter les proportions et le cadrage désiré. Cette épreuve est alors ajustée en largeur en **B** pour avoir un nombre de points multiple de huit. Ici sept octets soit 56 points. Puis en **C** on la transforme en équivalent noir et blanc. Jusqu'à ce stade la définition reste tout à fait acceptable. Mais l'afficheur

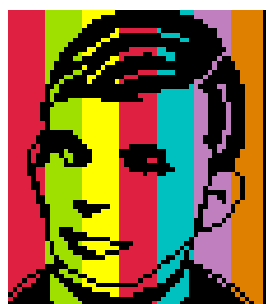


Fig.124

OLED ne gère pas des nuances de luminosité, c'est du tout ou rien. Il faut alors transformer en **D** ce dessin en BINAIRE. C'est cette phase qui dégrade le plus le portrait qui devient une triste caricature. Quelques retouches manuelles en **E** permettent de le rendre presque acceptable. On notera que la hauteur a été ajustée aux 64 lignes que peut restituer l'afficheur. On remarque également la bande noire sur la droite. Elle s'impose pour deux raisons : Cette colonne est obligatoirement contenue dans le "multiple de huit". Elle sera éteinte pour équilibrer les deux cotés du portrait et élargir un peu la zone de droite qui est limite pour afficher le texte souhaité. On repère

alors dans le dessin binaire les colonnes d'OCTETS comme sur la Fig.124 par des bandes de couleur. Puis sur la Fig.125 on retourne horizontalement chaque bande verticale car l'afficheur travaille sur des pages "imbriquées". Enfin,

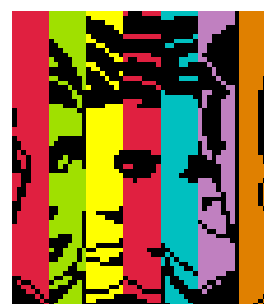


Fig.125

manuellement, ligne à ligne on analyse PIXEL par PIXEL pour coder les octets successifs, travail assez épouvantable qui justifierait pleinement l'usage d'un microcontrôleur pour l'automatiser. Sur le logiciel "définitif", la version du programme est obtenue avec le **BP3** dans le **MENU de BASE**. Par ailleurs le texte de la Fig.99 est actuellement remplacé par celui de la Fig.126 plus approprié. On constate en compilant `Arduino\P16_Machine_de_TURING.ino` que pratiquement 100% de l'espace dédié au programme est utilisé. *Il ne reste que quatre octets non utilisés* coté rentabilité on ne peut pas faire bien mieux ! Cher Alan ... on te devait bien ça !

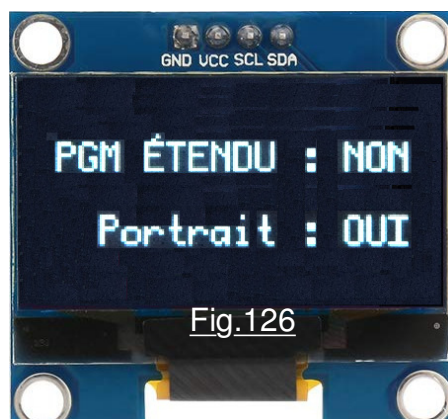


Fig.126

11) Une modification logicielle de dernière minute.

Expérimenter le programme de la machine autonome pour rédiger le petit manuel d'utilisation ainsi que le tutoriel d'apprentissage a fait émerger un point faible non négligeable dans le programme d'exploitation. Il semblait initialement qu'afficher le résumé des options au moment du **RUN** était une bonne idée ... qui s'avère rapidement très indigeste quand on est en train de mettre au point un algorithme. Durant cette phase on initialise les **OPTIONS**. Puis, chaque modification du programme est suivie d'un **RUN**. Avoir à cliquer une deuxième fois sur le **B.P.C.** pour sauter l'affichage des options devient à la fois lourd et inutile. Aussi, pouvoir suspendre cette possibilité est devenu indispensable. Hors le gaspillage de place pour afficher le portrait d'Alan **TURING** ne laissait que quatre octets en mémoire dédiée. Il a fallu faire des optimisations pour gagner une zone suffisante qui accepte la nouvelle option.

Ce sont encore les textes affichés qui ont été victimes et ont servi de "variable d'ajustement". ***Aussi, il ne faudra pas s'étonner d'observer ici et là quelques petits détails différents sur les écrans affichés, en particulier certains accentués qui ont été éliminés*** car relativement boulimiques en octets.

➤ Une option cachée.

Ajouter un Item dans le menu des **OPTIONS** était inenvisageable car les pages-écran sont saturées, ou la gestion d'un nouveau "poste" aurait été trop boulimique en octets de programme. Une autre stratégie a été adoptée, au détriment de la cohérence des menus. Ce n'est pas important, car à mon sens cette option cachée ne sera utilisée que rarement pour valider l'affichage du résumé des options qui n'est vraiment utile que dans des cas particuliers. Il semblait important lors du développement du logiciel, mais une fois que ce dernier est aboutit, on arrive assez rapidement à conclure que l'on peut oublier cette fonction. La solution adoptée pour résoudre le problème des



Fig.128

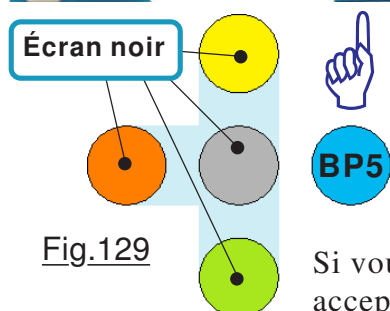


Fig.129

menus consiste comme le montre la Fig.127 à utiliser la page de présentation de la version du logiciel dans laquelle le texte **Machine de TURING** est remplacé par **Infos RUN** et à intervertir les deux lignes d'information. (Voir le résultat

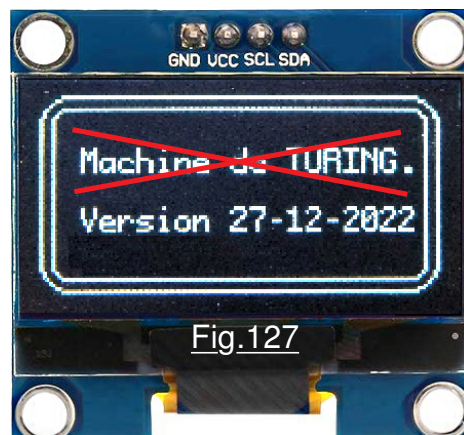


Fig.127

sur la Fig.128.) Comme cette page-écran est disponible dans le **MENU de BASE** avec le **BP5** elle sera facile à invoquer. Le protocole d'usage est élémentaire. En tournant le codeur rotatif on inverse **OUI** et **NON** à convenance. Quand on clique sur le **B.P.C.** il y a mémorisation de l'option en mémoire non volatile **EEPROM** puis retour au **MENU de BASE**. Noter que chaque fois que **BP5** sera cliqué dans le **MENU de BASE** il y aura une écriture en **EEPROM**, aussi il n'est pas recommandé d'abuser de cette fonction cachée.

Pour mémoire, j'insiste sur le fait que ce logiciel n'est accepté par la carte **ARDUINO NANO** que s'il ne dépasse pas les 30720 octets.

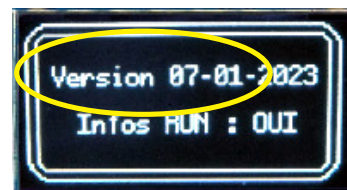
Si vous tentez de le transférer avec la version 1.7.9 la compilation ne sera pas acceptée. Il faut une version au moins égale à la 1.8.0 que j'utilise actuellement sous **WINDOWS VISTA**. Bien plus optimisée que les anciennes elle compacte le logiciel d'environ 10% ce qui est considérable quand on cherche désespérément de la place. Comme nous avons dilapidé à outrance dans les graphismes, et en particulier pour le portrait d'Alan **TURING**, on peut constater sur la Fig.130 qu'il ne aucun octet non utilisés en **SDRAM** programme. Coté rendement on reste parfaitement "crédible". Contrairement à l'alerte en orange du compilateur relative aux 502 octets restant pour les variables locales, rassurez-vous, c'est largement suffisant et il n'y aura aucun problème.

```
Le croquis utilise 30720 octets (100%) de l'espace de stockage de programmes
Les variables globales utilisent 1546 octets (75%) de mémoire dynamique,
ce qui laisse 502 octets pour les variables locales.
La mémoire disponible faible, des problèmes de stabilité pourraient survenir.
```

Fig.130

12) Avant de passer à la réalisation matérielle.

Développer un logiciel n'a rien à voir avec un processus linéaire. Au fur et à mesure de l'implantation des fonctions, le code enfle et le risque d'interférences entre les procédures "explose". On effectue de très nombreux tests avec une "campagne de vérifications" bien charnue. Puis, par la suite, une idée nouvelle surgit, et on améliore une routine sans se rendre compte qu'elle présente un "effet de bord" avec une autre fonction "éloignée". Par exemple dans le menu **BARILLET** on améliore l'affichage graphique du carrousel **INITIAL**. Satisfait du résultat, on passe à la suite, c'est à dire aux **OPTIONS**, car il y a bien longtemps que les fonctions du **RUN** sont au point. Mais il se trouve que l'exécution de l'algorithme utilise la même procédure d'affichage graphique à quelques variantes près. La modification dans "**BARILLET**" a introduit une vermine dans l'affichage en mode **RUN**. On ne s'en rendra compte que lorsque cette fonction sera à nouveau utilisée. Donc, tout semble fonctionner à la perfection. Du coup on passe au didacticiel et enfin au tutoriel. C'est la phase la plus prolifique en détection de problèmes. C'est précisément durant l'élaboration de **TUTORIEL.pdf** que surgissent les dernières vermines, car on passe en revue l'intégralité des fonctions et **OPTIONS**. Aussi, la probabilité qu'un ultime "bug" passe à la trappe est faible. Aussi, cette dernière facette du didacticiel a révélé des "misères" qui naturellement ont été corrigées. Aussi, il ne faudra pas vous étonner que la date de la version actuelle du programme difère de celle en Fig.3 du petit **MANUEL**. et que certains détails dans les affichages soient un peu différents des photographies du tutoriel.



➤ **Le programme P17_Machine_PERSONNELLE.ino.**

Durant la rédaction du tutoriel, en particulier en page 23 où une "erreur" de programmation nous a obligé à remplacer les **G** par des **D** et réciproquement l'idée de plus a surgi. C'est assez long à faire et sur la version MINIMALE une fonction s'en chargeait de façon automatique. Aussi, j'ai estimé que cette dernière était bien plus utile que celle qui permet de vérifier les lignes de programme NON utilisées. J'ai alors décidé de me faire un programme personnalisé dans lequel j'ai enlevé la fonction peu utile pour gagner de la place, et introduit dans **OPTIONS > Analyse** la nouvelle aide **Permuter G et D**. Il se trouve que cette dernière est nettement moins boulimique en octets que celle qui a été enlevée. Du coup j'ai ajouté à l'existant les diverses petites améliorations suivantes :

- Durant le mode **RUN** les affichages ont été simplifiés, car les multiples possibilités s'avéraient au final à la fois inutiles et peu commodes. Maintenant quand on tourne le bouton du codeur rotatif il n'y a plus que les trois options Graphique / GRILLE et Écran noir. Quand les multiples affichages ont été implémentés, je pensais que le temps pour afficher la page-écran était fonction du contenu alors qu'il est constant.
- Comme il restait beaucoup de place, j'ai dilapidé l'héritage. En particulier trois accents ont été ajoutés.
- **RAZ** du **BARILLET** retourne directement au **MENU de BASE**, car afficher la GRILLE obligeait à un clic de plus alors que l'information apportée n'était pas pertinente.
- Pour consommer des octets le petit cadre qui indique la référence de l'algorithme actuellement en mémoire a été doublé pour faire beau. (*Et oui, l'opulence pousse à l'orgie.*)
- Comme les 30720 octets n'étaient pas encore tous consommés, une dernière action boulimique a été tentée pour gaver l'ATméga328. Le **BP4** quand on est en **MENU de BASE** fait afficher la place de la mémoire dynamique qui reste entre la PILE et le TAS. C'est un outil pour les programmeurs qui sur notre machine n'apporte rien de plus à la qualité opérationnelle. Et pourtant ... Il reste encore 20 octets de non utilisés, et je n'ai plus d'idée pour les consommer !

Logiquement il serait sage de considérer que cette version **P17** soit celle officielle, mais il faudrait alors reprendre sérieusement le didacticiel et le tutoriel. Franchement je n'en ai pas le courage, prétextant que cette version ultime ne fait qu'ajouter une nouvelle fonction facile à utiliser. Il suffit de l'invoquer, de tourner le codeur rotatif pour avoir à votre choix **OUI** ou **NON** et de cliquer deux fois sur ce dernier pour revenir au **MENU de BASE**. En conclusion, je vous suggère de commencer par téléverser le programme "officiel" **P16**, de faire toutes les expériences du **TUTORIEL**. Puis, vous téléversez la version **P17**, vous la testez et enfin vous décidez de la conserver ou de reprendre **P16**.

Quoi qu'il en soit, je souhaite que ce didacticiel et ce tutoriel sur la Machine AUTONOME vous aura séduit et pourquoi pas vous incitera à franchir le pas. Bien que ce soit déjà précisé dans plusieurs documents, si vous rencontrez un problème : michel.droui@laposte.net

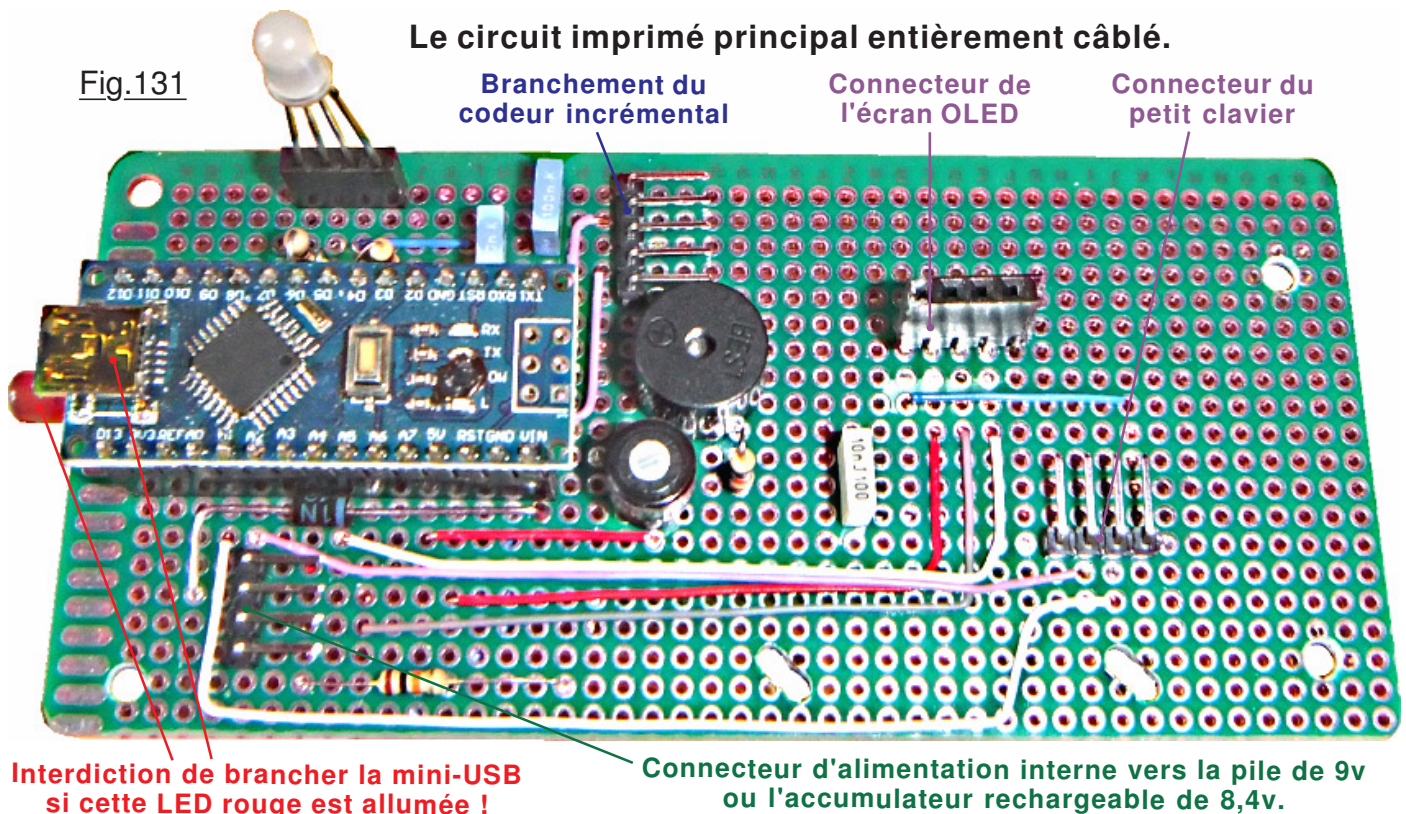
13) Concrétiser cette version autonome de la machine "Arduino".

C'est à double titre que cette ultime version de machines de Turing est autonome. D'une part elle n'a plus besoin d'être reliée à un ordinateur pour établir le dialogue Homme/Machine. Par ailleurs, une alimentation "locale" évite d'avoir "le fil à la patte" au point de vue énergétique. Bien qu'il soit possible d'alimenter à partir du secteur avec un adaptateur USB via la mini prise, il est également prévu d'employer une source d'énergie propre qui sera à votre choix une pile 9v GLR61 ou un accumulateur rechargeable de 8,4v de type Ni-MH. Toutefois, cette double possibilité n'est pas sans risque pour la carte Arduino si l'utilisateur effectue une mauvaise manœuvre, il faudra faire bien attention.

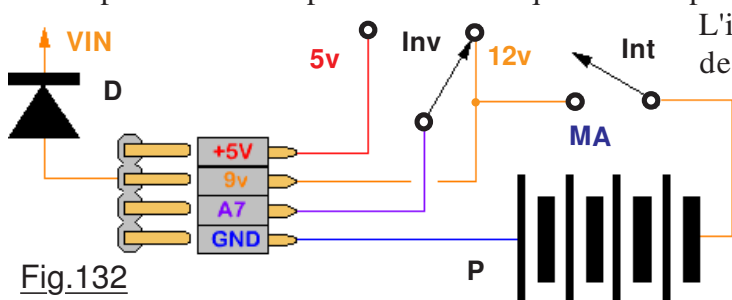
➤ **Interdiction de brancher la mini-USB si l'alimentation interne est activée.**

Stipulé de façon formelle dans l'encadré rouge de la "notice" d'utilisation de la carte Arduino NANO donnée en Fig.133, il ne faut strictement pas brancher à la fois une tension sur **VIN** et du 5Vcc par l'entremise de la mini-prise USB sous peine de détruire le régulateur 5v local qui à parti de la broche **VIN** alimente la carte en 5v continu et stabilisé.

NOTE : Quand on alimente la carte Arduino par sa mini prise USB, la tension positive du +5Vcc se retrouve sur la broche **VIN**. Il est donc normal que la LED rouge qui signale que l'accumulateur ou la pile 9v est en ligne s'illumine. Son but n'est pas de signaler la présence de l'énergie **mais d'interdire le branchement de la fiche mini USB si elle est allumée.**



À partir du moment où l'appareil électronique est pourvu d'une alimentation "embarquée", on se doute qu'il faut pouvoir librement la mettre en service ou la couper. Le schéma électrique relatif à la gestion de cette alimentation interne est proposé en Fig.132 et ne comporte qu'un inverseur **Inv** est un interrupteur **Int**. C'est précisément **Int** qu'il ne faut pas fermer sur **MA** lorsque la mini-USB est branchée.



L'inverseur **Inv** permet de sélectionner pour la mesure de tension en entrée **A7** le +5Vcc stabilisé, ou la tension aux bornes de la pile **P**. (Ou de la batterie rechargeable.) Si par mégarde l'utilisateur introduit la pile dans le mauvais sens dans son compartiment, la diode **D** protège le régulateur contre une inversion de polarité.

Suite en page 43.

14) 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 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.)

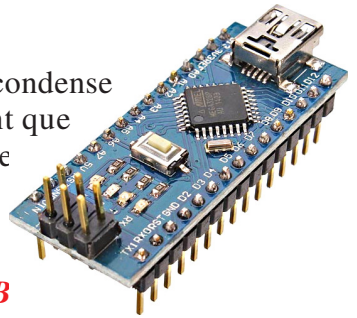
8 broches d'entrées Analogiques dont 6 pouvant fonctionner en E/S.

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

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

L'ATmega328 a 2 Ko de SRAM 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)`.

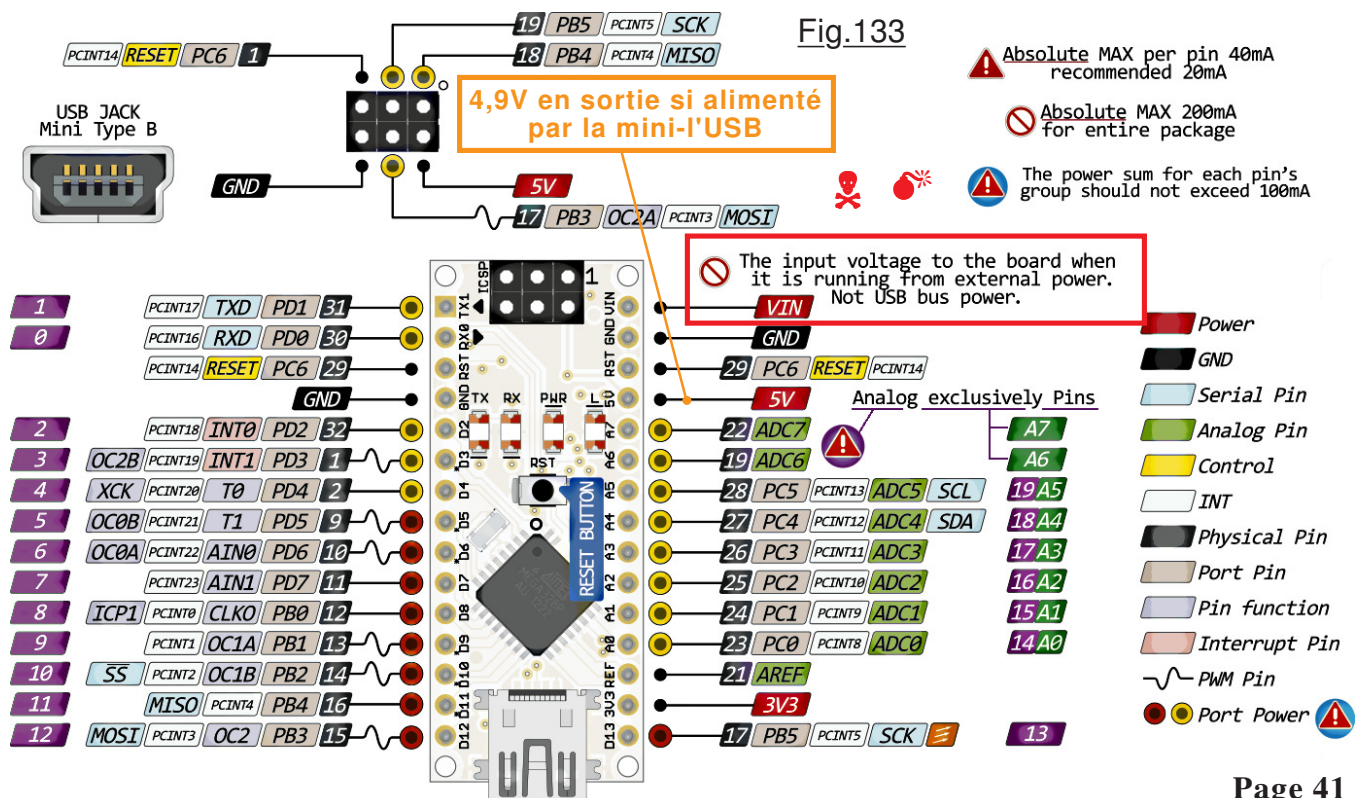


De nombreux clones chinois de ces cartes existent. Certains sont basés sur des circuits intégrés tels que l'ATmega328P CH340G qui ne sont pas reconnus directement sous Windows sans installer un pilote spécifique. Pour mes cartes le driver adapté a été trouvé sur :

<http://www.mediafire.com/download/pjqn88uc64acpgz/ch341ser.zip>

L'exécutable une fois activé la ligne USB fonctionne normalement et les NANO se programment sans problème à condition :

- De sélectionner le type **Arduino Nano** avec **Outils ...**
- D'activer le bon port pour la ligne USB.



➤ Le circuit imprimé principal.

Étudié pour supporter la grande majorité des composants, il est réalisé à partir d'un morceau de carte imprimée préperçée adaptée au prototypage. Cet élément mesure 120mm x 55mm. Diverses photographies commentées sont rangées dans le dossier <Galerie d'images\Machine AUTONOME>. Sur [Image 17.JPG](#) on a commencé par souder les composants les moins hauts pour finir par les connecteurs et le petit bruiteur. L'[Image 18.JPG](#) présente le circuit terminé. C'est la hauteur du petit clavier qui conditionne la position de la façade supérieure du boîtier. Comme le montre [Image 22.JPG](#) les deux circuits imprimés partagent deux collones de fixation sur la semelle de coffret. Du coup la hauteur du clavier détaillé sur [Image 20.JPG](#) impose celle de la LED triple. Pour surélever cette dernière, on peut observer sur [Image 18.JPG](#) et [Image 19.JPG](#) qu'elle est simplement enfichée sur un support femelle de type HE14. En particulier sur [Image 19.JPG](#) on insiste sur l'interdiction de brancher la fiche mini-USB si la LED rouge est allumée, et surtout on prévient qu'au montage il faudra se montrer particulièrement attentif quand on va assembler l'afficheur OLED. Ce dernier est soutenu par son connecteur HE14 qui est particulier. Ses

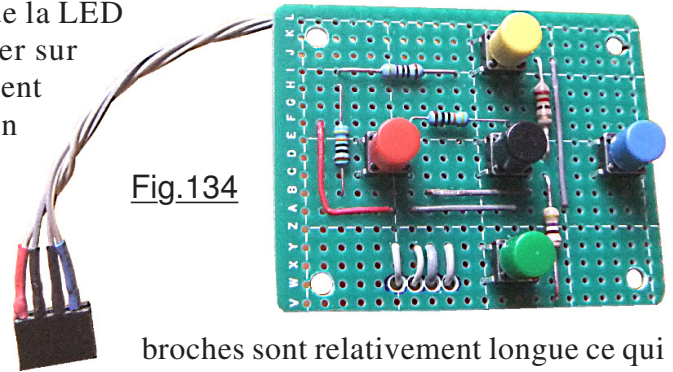


Fig.134

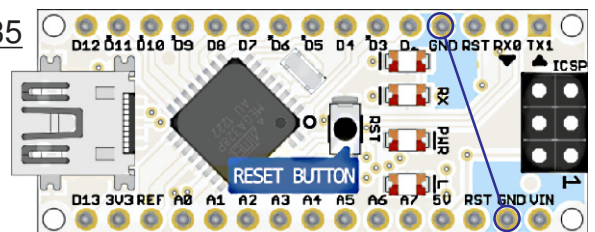
broches sont relativement longue ce qui Quand on immobilise ce petit écran avec des entretoises et des petits boulons ϕ M2 il importe de faire très attention à la plaque de verre de surface particulièrement fragile. À peine vous la brisez et l'afficheur peut servir de porte-clef ! Je vous conseille fortement de brancher le connecteur du petit clavier (*Fiche HE34 femelle de la Fig.134*) sur le circuit imprimé principal **avant d'installer l'afficheur**, ce que montre [Image 21.JPG](#). La photographie d'[Image 23.JPG](#) montre bien le connecteur HE14 femelle qui supporte l'afficheur. Manifestement ses broches sont plus longues qu'en standard, et il n'est pas soudé contre la plaque cuivrée. Rassurez-vous, on constate facilement sur [Image 24.JPG](#) qu'il n'y a pas beaucoup de liaisons à réaliser sous les deux circuits imprimés. Sur [Image 25.JPG](#) l'ensemble est vu du côté de la LED triple. Enfin [Image 26.JPG](#) et [Image 27.JPG](#) montrent les deux circuits assemblés pour effectuer leur validation. Dans ce but le codeur incrémental a été branché provisoirement pour effectuer les essais. C'est à ce stade que l'on doit faire appel à l'outil logiciel <OUTILS du programmeur\Calibrage_des_CAN.ino> pour mesurer les tensions présentes lors de l'activation des touches et optimiser les seuils de comparaison dans le programme d'exploitation. Noter que les valeurs indiquées sur Fig.21 à Fig.23 du didacticiel sont issues de ce test final. Toutefois, avant d'enficher la carte Arduino NANO sur ses deux rampes HE14 et de l'alimenter il me semble incontournable d'effectuer une vérification méthodique et complète du circuit imprimé de base.

➤ Vérification complète du circuit imprimé principal.

Première vérification impérative, se munir d'une loupe à fort grossissement, nommée "compte fils" et vérifier visuellement chaque soudure et s'assurer qu'elle ne déborde pas accidentellement sur un élément voisin. (*Pastille, fil de liaison, broche d'un HE14 ...*) On doit ensuite examiner les nombreuses liaisons et leurs isollements avec un multimètre, mais franchement un **TESTEUR DE CONTINUITÉ** sera bien plus convivial car on n'a pas à regarder un cadran. **La validation commence toujours sans la présence de la carte Arduino NANO.**

- 1) Vérifier l'extrémité de chaque ligne conductrice en pointant la broche du connecteur concerné et la lyre associée sur le support de la carte Arduino NANO. Par exemple le +5Vcc sur le connecteur fournissant l'alimentation régulée, et la broche notée **5V**. On peut aussi tester entre **GND** sur le connecteur fournissant l'alimentation régulée, et la broche **GND** d'Arduino. ATTENTION, il y a deux broches **GND**, une seule va au connecteur d'alimentation. L'autre est pontée par le circuit imprimé. (*Voir la Fig.135*)
- 2) Quand l'intégralité des liaisons a été vérifiée ainsi que tous les isollements de voisinage, brancher du +5V et **GND** sur le connecteur HE14 à quatre broches dédié à l'alimentation en interne par pile ou par accumulateur. **La consommation doit être pratiquement nulle.**

Fig.135



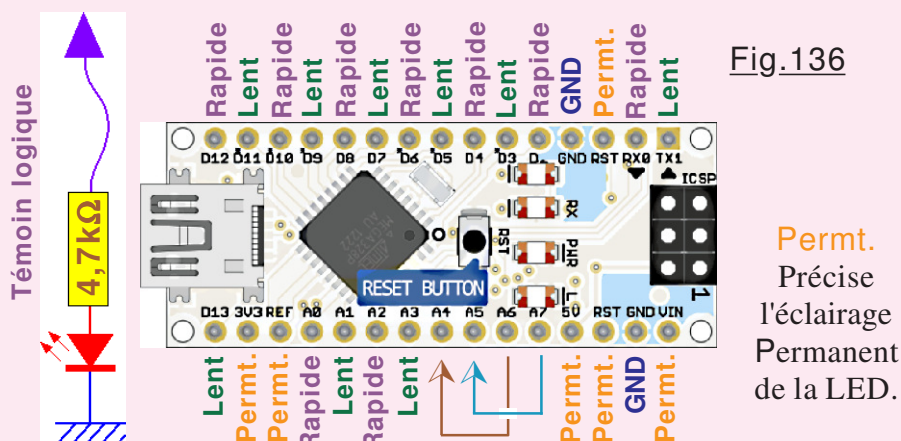
- 3) Ponter le **+5Vcc** sur la broche correspondant à **VIN**, la LED rouge doit s'allumer.
- 4) Tester ensuite avec le **+5Vcc** sur les lyres correspondant à **D5**, **D6** et **D7**. Les composantes rouge, verte et bleue de la LED triple doivent s'allumer.
- 5) Ponter ensuite le **+5Vcc** sur la lyre du microcontrôleur qui correspond à la sortie binaire **D4**. Le Buzzer doit s'activer et générer un BIP franc. Si la tonalité n'est pas nette, c'est certainement que la résistance insérée fait plus de **100Ω**.
- 6) Couper l'alimentation. Brancher le petit clavier et immobiliser toutes les entretoises sur les deux plaquettes cuivrées comme montré sur [Image 22.JPG](#) pour aboutir à un groupe compact et stable.
- 7) Rétablir le +5vcc et tester les tensions issues du clavier sur les broches relatives à **A0** et à **A1**.
- 8) Couper l'alimentation et introduire l'afficheur OLED sur son support. Mettre en place ses deux entretoises en prenant les précautions indispensables pour ne pas l'excorier. Noter au passage sur [Image 21.JPG](#) que les deux trous de passage des vis sont en diagonale et allongés pour faciliter la coïncidence entre les entretoises et les deux trous de fixation de l'afficheur.
- 9) Quand on a effectué une vérification complète du circuit imprimé et que l'on est certain que tout est conforme au schéma, mettre en place la carte Arduino NANO supposée opérationnelle, c'est à dire avec les données en EEPROM et le programme d'exploitation **P17** téléversé. Brancher la mini-fiche à un petit adaptateur USB secteur. Une ou deux secondes après la mise sous tension, l'afficheur OLED doit immédiatement présenter le portrait d'Alan Turing. *Si ce n'est pas le cas couper immédiatement l'alimentation et cherchez l'erreur de câblage.*
- 10) C'est à ce stade que l'on téléverse l'outil logiciel [Calibrage_des_CAN.ino](#) pour mesurer les valeurs typiques produites par clavier réalisé avec vos propres composants. Puis vous optimisez les constantes de comparaison dans **P17** qui est enfin téléversé et réputé "définitif".

Valider une carte NANO.

Ant de procéder à la validation des circuits imprimés, opération décrite en détails dans le chapitre qui précède, il me semble logique de nous assurer que la carte Arduino NANO utilisée est totalement opérationnelle. Ce test initial peut se faire avec l'outil logiciel **Tester_la_carte_NANO.ino** qui n'impose en réalité strictement aucun circuit imprimé spécifique, car on peut brancher les fils d'expérimentation directement sur les broches de liaison du petit circuit imprimé. Pour téléverser le croquis, on branche la ligne Mini-USB, le minuscule circuit électronique est alors alimenté en 4,4Vcc. Déjà sans rien connecter deux LEDs du module clignotent alternativement. Pour conduire les vérifications, on improvise le petit témoin logique de la Fig.136.

Les 14 sorties binaires clignotent à une cadence d'environ 0,8S. Pour pouvoir s'assurer que deux broches voisines ne sont pas en contact intempestif, les sorties paires génèrent des impulsions lumineuses très courtes, alors que les sorties impaires s'illuminent à peine durant 0,21 seconde. Les broches d'entrées analogiques programmées en sorties **A0**, **A1**, **A2** et **A3** fonctionnent de la même façon. Pour vérifier les deux entrées Analogiques **A6** et **A7** qui ne peuvent pas fonctionner en sorties, elles "recopient" leur état respectivement sur **A4** pour l'entrée **A6** et sur **A5** pour l'entrée **A7**. Pour tester ces deux entrées, il suffit de les relier soit à **GND**, soit au **+5Vcc** alternativement. Le témoin logique branché sur la sortie associée sera éteint ou allumé. (À la temporisation près.)

Résumé du comportement attendu par le programme :



15) Coffret de la machine autonome avec batterie 9v interne.

Phase ultime de ce projet alléchant, elle conditionne directement le plaisir que l'on aura à utiliser ce petit module électronique. Aussi, soigner son esthétique et sa compacité sont les critères principaux de la conception du boîtier. Chacune et chacun va privilégier ses techniques personnelles. Pour ma part, je vais illustrer ce propos avec un boîtier conçu en polystyrène choc, matériau dont je me sers depuis des années et des années. Avant de passer aux détails de réalisation du coffret, notez que les dessins des circuits imprimés sont donnés dans le fichier [EXCLUSIONS.pdf](#) qui intègre deux fiches au format A5. Ce document doit être imprimé Recto/Verso et pour ma part mes fiches sont plastifiées.

➤ **Le branchement de la pile ou de la batterie rechargeable.**

Suite à diverses études, au final il a été décidé de loger la pile 9v sous le clavier. La zone était inutilisée, on optimise l'utilisation du volume du coffret et on en minimise l'encombrement. Comme coté clavier le circuit imprimé principal n'était pas encombré, j'ai décidé d'y solidariser les deux contacts électriques flexibles sur lesquels viendra buter l'accumulateur. L'ensemble des éléments assurant cette liaison électrique et montré sur la Fig.137 est issu des lamelles d'un ancien relais électromagnétique. Ces deux pièces de cuivre ont fait l'objet d'un travail "d'horloger" pour les percer au pas de 2,54mm et pour leur donner la forme favorisant l'élasticité en flexion. Toutes ces pièces sont très petites, puisque la visserie est en ϕ M2. L'agencement de ces petits éléments est détaillé et commenté sur les photographies [Image 28.JPG](#) à [Image 34.JPG](#) dans [<Galerie d'images\Machine AUTONOME>](#). Il me semble assez évident que nombreuses et nombreux seront celles et ceux qui ne disposeront pas de lamelles de cuivre de récupération pour créer comme sur la Fig.137 les deux plots de contact. Aussi, il sera alors judicieux d'acheter dans le commerce en ligne un petit réceptacle de pile GLR61, quitte à revoir un peu les dimensions du coffret pour le loger sur le coté s'il ne passe pas entre les colonnes du clavier ce qui me semble inexorable.

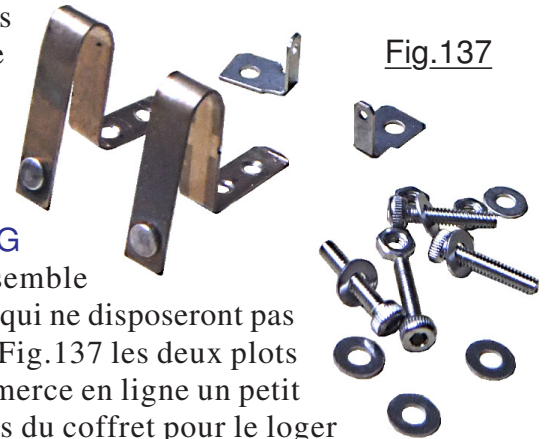


Fig.137

➤ **L'alvéole de logement de la pile ou de la batterie rechargeable.**

Intégré directement sur la semelle du coffret la Fig.138 présente les divers éléments assemblés sur cette dernière. La réalisation de ce petit compartiment est largement détaillée sur les photographies [Image 35.JPG](#) à [Image 42.JPG](#) également disponibles dans le dossier [<Machine AUTONOME\L'alvéole pour contenir la pile>](#). Comme ce chapitre fait appel à l'usinage et au soudage de plusieurs pièces en polystyrène choc, je vous invite fortement à consulter le document intitulé [Réaliser un COFFRET.pdf](#) qui accompagne ce didacticiel. Notez au passage que le petit fil rigide rouge qui sur [Image 32.JPG](#) servait à tenir vers le haut la ligne filaire du clavier n'est plus du tout utile. On le constate tout particulièrement sur [Image 39.JPG](#) et [Image 40.JPG](#) alors qu'[Image 42.JPG](#) analogue à celle de la Fig.138 présente l'ensemble sous tension par l'adaptateur USB.

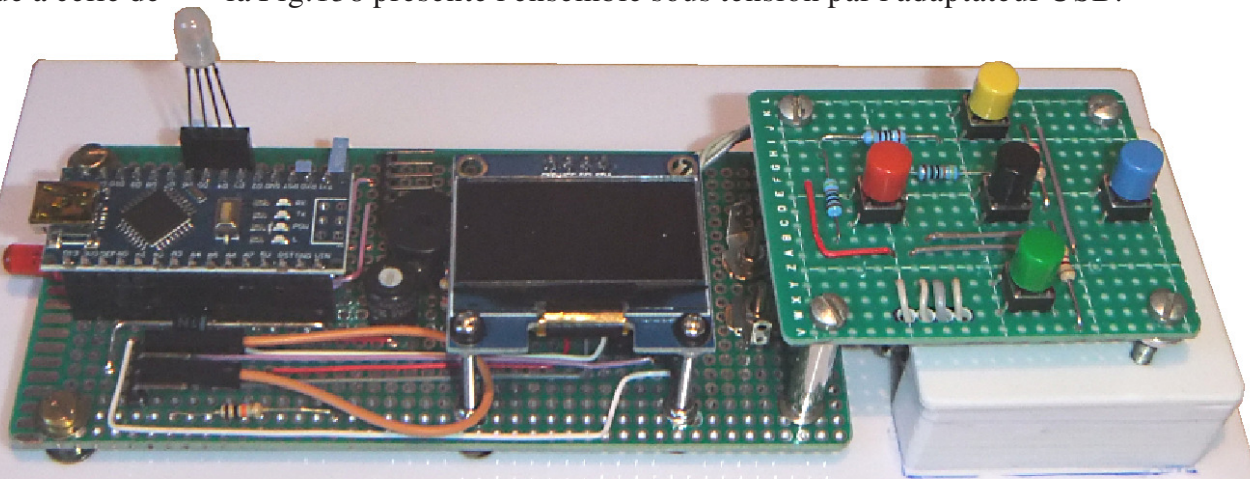


Fig.138

16) Étude informatique du coffret.

Habituellement, pour réaliser un boîtier aussi simple que celui de notre petite machine, je me contente de réaliser deux ou trois dessins à l'échelle unitaire. Puis, quand le boîtier est entièrement défini, je passe à la réalisation pratique avec les techniques de travail du polystyrène choc décrites dans le document [Réaliser un COFFRET.pdf](#) joint à ce didacticiel. Mais nous sommes en hiver, l'atelier est bien trop froid pour y travailler avec précision. Aussi, disposant de temps, j'ai pensé que réaliser une étude précise sur l'ordinateur pourrait s'avérer utile et ainsi permettrait de vous proposer des dessins précis.

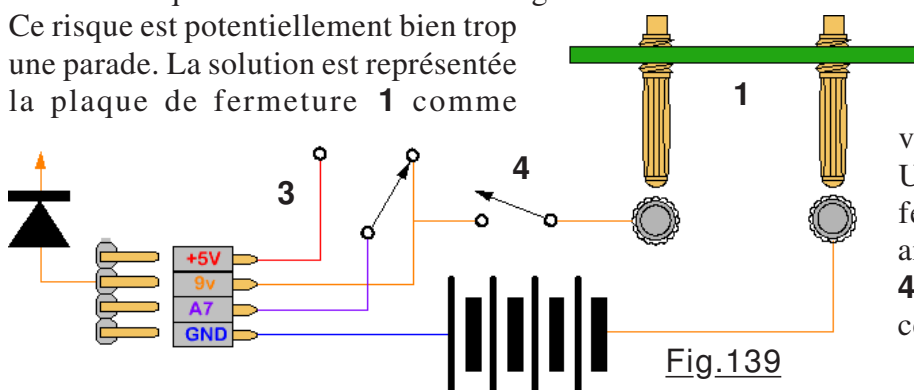
➤ Changement de morphologie.

L'organisation générale du boîtier présentée sur la Fig.18 a été modifiée pour l'optimiser en vue de son exploitation par un opérateur droitier, catégorie à laquelle j'appartiens. L'expérience m'a montré qu'il était plus commode de placer le codeur rotatif à gauche et le clavier à droite. Sur le plan de la convivialité la différence n'est pas très importante, mais comme l'on réalise sur mesure, autant pousser au maximum la qualité opérationnelle. Je suggère du coup aux personnes gauchères d'adopter une configuration symétrique ce qui obligera à revoir l'implantation de l'ensemble. La Fig.138 montre l'allure générale que présentera l'appareil définitif, avec le codeur rotatif en **9** et le petit clavier multicolore en **7**. En **5** une petite "vitre" en matière thermoplastique translucide ferme l'ouverture de l'afficheur OLED. Entre le coffret et cette dernière une petite pièce en papier imprimée portera les informations relatives aux fonctions attribuées aux deux inverseurs **3** et **4**. L'inverseur **3** sélectionne la source de tension mesurée, quand à **4** c'est l'inverseur Marche / Arrêt lorsque la petite unité fonctionne en autonome sur sa batterie rechargeable interne. En **6** on peut observer la plaque de fermeture qui permet de la sortir pour la recharger, ou simplement la changer si c'est une pile ordinaire de 9v. En **2** la diode électroluminescente tricolore dépasse sur le dessus, alors que l'orifice **8** autorise le passage d'un stylet pour provoquer un RESET sur la carte Arduino NANO quand elle est en phase de développement de programme. Enfin en **1** une plaque de fermeture autorise ou interdit de brancher la mini fiche d'un cordon qui permet de l'alimenter par un adaptateur secteur USB ou de la faire dialoguer avec un P.C. pour développer le programme.

➤ Changement de stratégie.

Supposons que l'on est en configuration de programmation, ou que l'appareil soit alimenté par un adaptateur secteur USB. Naturellement, avant de brancher ce dernier on a placé l'inverseur **4** sur Arrêt, car nous savons qu'il est interdit d'utiliser **VIN** si la mini prise USB est alimentée en énergie. Ce problème est décrit dans le chapitre 13 en page 41. Par mégarde, au lieu de basculer l'inverseur **3** pour changer la source de tension mesurée on se trompe et on place **4** sur sa position Marche. Dans la fraction de seconde qui suit on aura détruit le régulateur interne de tension. Dommage !

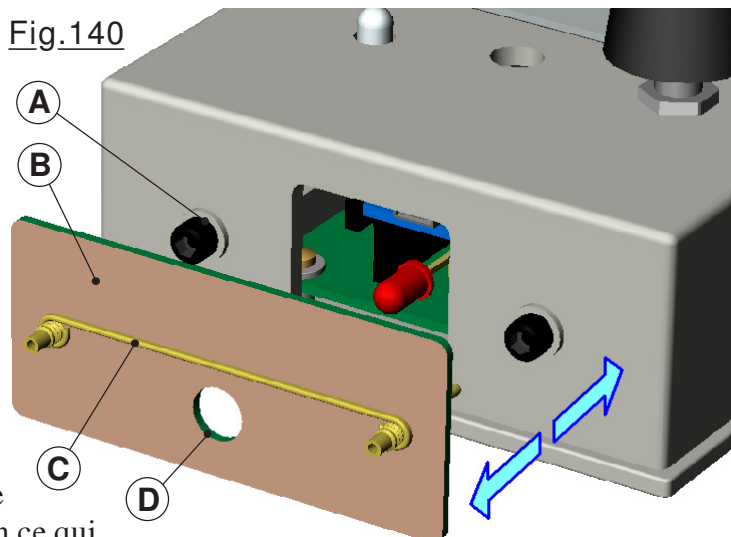
Ce risque est potentiellement bien trop une parade. La solution est représentée la plaque de fermeture **1** comme



important pour ne pas y apporter sur la Fig.139 et consiste à utiliser "coupe circuit" électrique. Si on veut alimenter avec la petite prise USB, il faut enlever la plaque de fermeture **1**. Comme c'est elle qui amène le +9V de la pile sur l'inverseur **4**, on la débranche ainsi de façon certaine et automatique.

La Fig.140 ci-contre présente la solution envisagée informatiquement. Sur le boîtier sont prévues deux prises **A** pour fiches bananes de deux millimètre nominal. ces fiches sont soudées coté cuivre sur une petite plaque de circuit-imprimé **B**. Pour parfaire la conductivité et la rigidité mécanique, un fil de cuivre **C** est soudé et cimplète l'ensemble. Un orifice **D** est pratiqué sur la plaquette cuivrée pour laisser visible la diode électroluminescente rouge. On peut ainsi vérifier que la carte Arduino NANO est bien alimentée par la pile lorsque l'inverseur **4** est sur la position Marche. C'est simple, facile à mettre en œuvre et surtout assure une sécurité absolue en ce qui concerne les erreurs de manipulation que peut engendrer l'utilisateur.

Fig.140



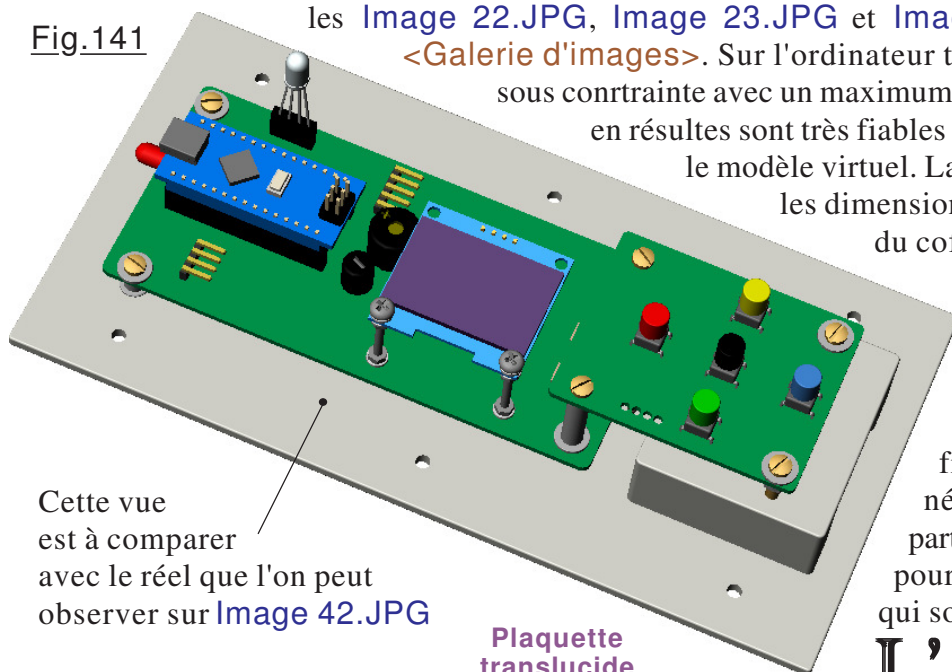
> L'agencement de la semelle.

Simple plaque en polystyrène choc de 3mm d'épaisseur, cette dernière sert de support aux deux circuits imprimés de l'appareil et constitue le logement de la pile d'alimentation. À rapprocher avec les [Image 22.JPG](#), [Image 23.JPG](#) et [Image 40.JPG](#) préservées dans la

Fig.141

<Galerie d'images>. Sur l'ordinateur tous les composants sont réalisés sous contrainte avec un maximum de précision. Ainsi les dessins qui en résultes sont très fiables pour extraire la cotation et valider le modèle virtuel. La semelle a été dessinée en prenant

les dimensions sur le réel. En revanche le reste du coffret a été entièrement développé en virtuel. Puis, lorsque la vérification de non-interférence et les écarts entre les différent éléments ont été confirmés, alors le modèle a été considéré comme fiable pour déterminer les éléments nécessaires à la réalisation. Tout particulièrement la position et les cotes pour toutes les lumières et les orifices qui sont sur le dessus.



Cette vue est à comparer avec le réel que l'on peut observer sur [Image 42.JPG](#)

Plaque translucide

Codeur rotatif

L'avantage de l'informatique, c'est que sur un dessin d'ensemble on peut faire disparaître à notre guise certains éléments et les autres restent "en l'air" à leur position comme par magie. C'est le cas sur la Fig.142 ci-contre qui est particulièrement utile pour que le concepteur puisse vérifier qu'il n'y a pas d'interférence matérielle et *que la distance entre tous les éléments est suffisante pour tolérer à la fabrication des marges d'imprécision de façonnage raisonnables.*

Fig.142

La hauteur du clavier est très importante car le boîtier fait 3mm d'épaisseur et les boutons poussoir sont très petits. Hors il faut qu'ils dépassent suffisamment de la face supérieure du coffret.

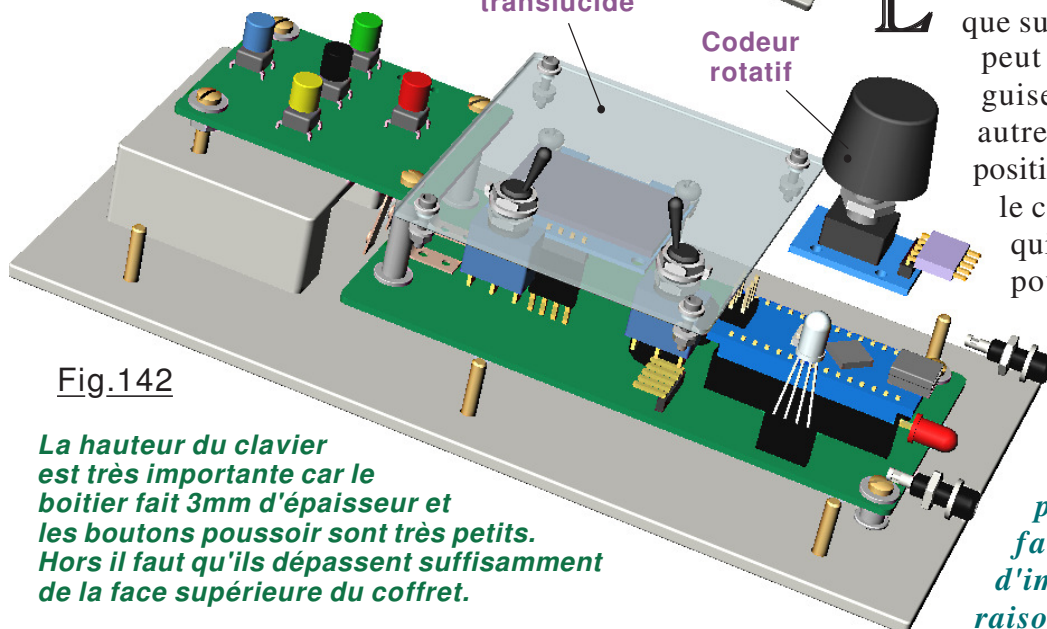
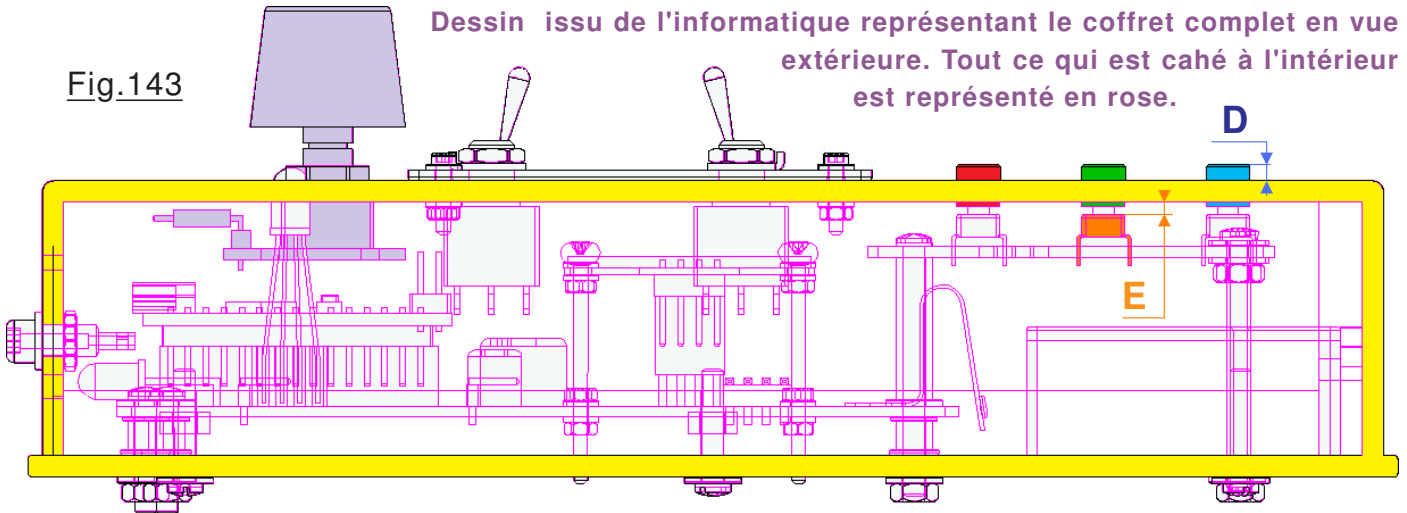


Fig.143

Dessin issu de l'informatique représentant le coffret complet en vue extérieure. Tout ce qui est caché à l'intérieur est représenté en rose.



Profitant des facilités de l'informatique, on peut bénéficier de vues précises comme sur la Fig143 l'ensemble étant montré en vue extérieure, mais avec le tracé de tout ce qui se trouve à l'intérieur ici en rose. On peut vraiment s'assurer qu'il y a bien de l'espace entre tous les éléments. Et surtout, il devient possible d'optimiser les dimensions. Par exemple la hauteur du boîtier a été minimisée en s'assurant que les boutons du clavier dépassent suffisamment en **D**, et surtout que le corps des touches ne talonne pas et laisse un espace d'environ 1mm en **E**. Issus du modèle informatique et réunis dans le fichier joint à ce didacticiel [Dessins du coffret.pdf](#) vous trouverez les croquis cotés des pièces principales.

17) Réalisation matérielle du coffret.

Étape ultime du projet, la matérialisation du boîtier décrite ici suppose qu'il sera composé d'éléments en polystyrène choc dont les techniques d'usinage sont décrites dans le document [Réaliser un COFFRET.pdf](#) déjà mentionné en page 46. La première phase consiste à créer la semelle avec le logement de l'accumulateur rechargeable. Mis à part les six trous de passage des vis de liaison avec le boîtier, l'ensemble des orifices de passage des vis d'assemblages est concrétisé, ainsi que les deux trous taraudés sur le logement de la pile traversé par les vis longues qui supportent le petit clavier. La photographie d'[Image 42.JPG](#) présente la semelle dont les dimensions sont légèrement plus grandes que celles désirées. Lorsque le boîtier sera terminé et que les six vis d'assemblage de la semelle seront percés, on assemblera le tout qui sera parfaitement positionné par rapport aux éléments intérieurs qui traversent sur le dessus. On tracera alors le contour définitif en débordement d'environ 1mm et la semelle sera définitivement ajustée aux bonnes dimensions. La deuxième phase consiste à réaliser les cinq éléments qui soudés les uns aux autres vont constituer la surface latérale et le dessus du boîtier. Dans le dossier de photographies commentées

État	Lectr	Ecrit	Mvt	ÉTAT
1	1	0	D	T5
1	B:	1	D	T2
2	0:	0		
2	B:	D		T3
3	0:	1	D	T4
3	1:	0		
4	B:	D		T1
4	1:	0		
5	0:	0		
5	B:	D		T6
6	1:	D		T5
6	B:	114 cycles d'horloge		

<Réaliser le coffret> [Image 43.JPG](#) présente la face coté accumulateur en cours de façonnage. Les alvéoles pour les écrous prisonniers sont terminées. Il faut encore faire les deux plaques qui soudées sur les cotés empêcheront les écrous de sortir. À ce stade il n'est pas impératif d'ajuster finement la partie inférieure qui sera vers l'extérieur. Elle sera peaufinée à la fin quand le boîtier sera complètement assemblé. C'est à ce stade que l'on ajustera toute la surface sur une feuille de papier abrasif. Sur [Image 44.JPG](#) on peut observer l'autre face latérale qui supporte les deux douilles pour fiches bananes $\phi 2\text{mm}$. L'une de ces fiches est en place sur cette vue. Durant l'usinage des diverses pièces on les

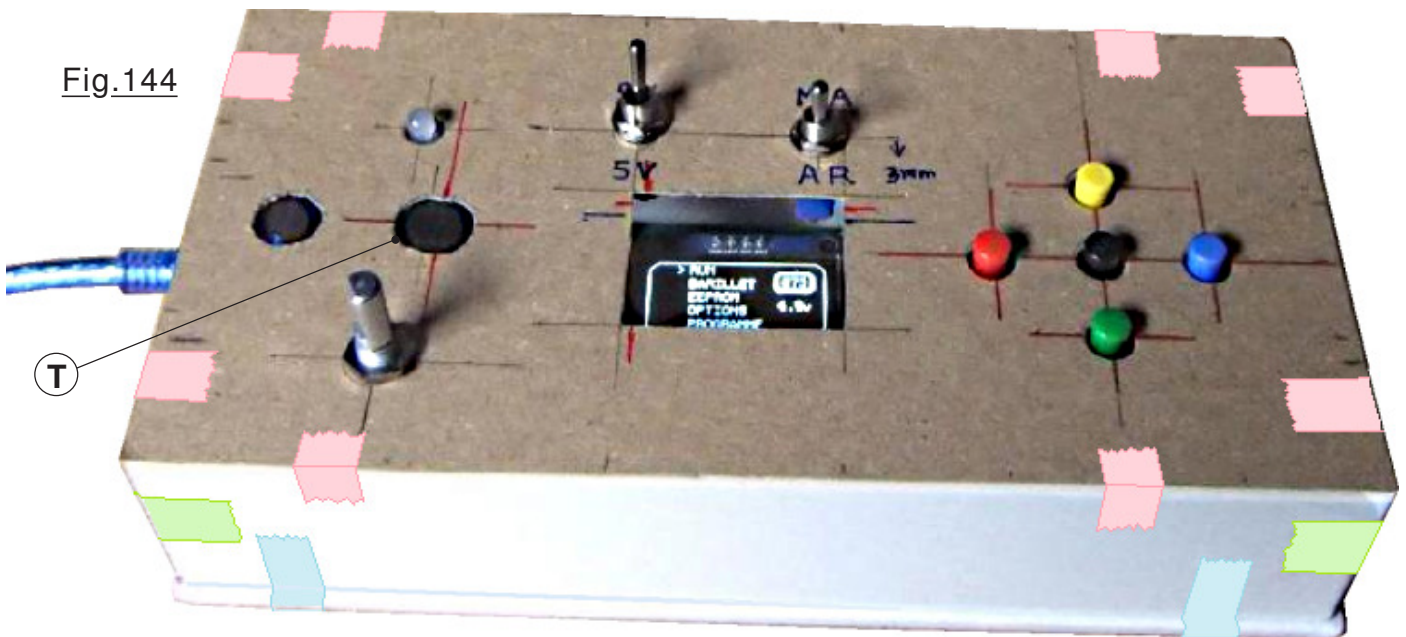
En page 52 je vous propose un petit exercice de programmation. Ici vous trouverez ma solution. Le tableau est sens dessus dessous pour ne pas qu'il soit trop facile de le lire, notamment sans vraiment vouloir le faire.

manipule dans tous les sens. Il devient alors très risqué d'inverser une position. Hors ces éléments ne sont pas strictement identiques. Aussi, dès que les débits sont réalisés et affinés à leurs dimensions définitives, il faut, comme sur [Image 45.JPG](#) les repérer de façon à ne plus les intervertir et les inverser en orientation. **L'une des opérations les plus délicates consiste à percer les trous et la lumière rectangulaire sur la plaque qui deviendra le dessus du boîtier.** Il faut que tous les orifices soient exactement dans l'alignement ou centrés sur les éléments situés à l'intérieur, (*Bouton de RESET, écran OLED ...*) et dans l'axe les boutons poussoir du clavier. Pour relever ce défi, voici la technique que je vous propose :

➤ Un patron rigoureux pour assurer l'entreprise !

Cette approche suppose que la semelle est entièrement achevée et que tous les éléments qu'elle supporte sont en place définitive sur cette dernière. On commence par découper une pièce en carton rigide réalisées aux dimensions exactes de la plaque du dessus en polystyrène choc. Puis, avec un maximum de soin on trace sur cette dernière la position théorique de tous les trous. Avec une paire de ciseaux bien pointus ou un cutter on dégage tous les trous. À ce stade ce n'est pas important qu'ils soient parfaitement positionnés. Quitte à agrandir certains, il faut impérativement que le carton puisse se placer sur les faces latérales sans être en interférence avec les éléments de la semelle. Par exemple, on constate sur la Fig.144 que le trou **T** au dessus du bouton de RESET a été allongé pour pouvoir tracer en rouge les axes de positionnement définitif de cet orifice. On assemble alors les pièces latérales et le dessus en carton avec du ruban adhésif en veillant à ce que les faces latérales soient bien en contact entre elles et avec la semelle. Sur la Fig.144 l'opération de pointage précis des positions est en cours. Comme la photographie est un peu surexposée et que le ruban adhésif y est "invisible", l'image à été caricaturée manuellement pour bien situer les morceaux translucides qui assurent la cohésion de l'ensemble. On trace

Fig.144



alors définitivement les axes de position des divers orifices. Par exemple il a été décidé de décaler de 3mm les deux inverseurs vers l'opérateur. Il a également été prévu, ce que montre [Image 48.JPG](#), de diminuer sur "le haut" la taille de la lumière d'observation du petit écran OLED. [Image 46.JPG](#) et [Image 47.JPG](#) présentent des détails de l'opération en cours. Dernière phase de cette étape, on immobilise parfaitement le carton sur la pièce définitive, ce que montre [Image 49.JPG](#) et on y porte avec précision les centres des trous et les contours de la lumière rectangulaire ménagée pour voir l'afficheur OLED.

Préparer la face supérieure consiste alors à réaliser l'intégralité des orifices. *On les perce tous à un diamètre inférieur à celui désiré.* Puis, en utilisant l'alésoir de la Fig.9 en page 5 du document [Réaliser un coffret.pdf](#) on affine ces trous de façon à ce qui doit les traverser pénétrer "en sifflant", c'est à dire pratiquement sans jeu. Pour les touches du clavier même punition. On doit pouvoir placer la plaque avec pratiquement aucun espace. Ce n'est que tout à la fin, quand le boîtier complet sera immobilisé sur la semelle que l'on agrandira "le juste ce qu'il faut" pour que les touches soient bien centrées avec un jeu maximal d'un millimètre. (*Plus ce jeu sera faible, plus belle sera la façade.*) Un dernier petit conseil : De la pleine réussite de cette opération dépend directement l'esthétique du boîtier. Aussi, soignez un maximum votre travail et prenez votre temps. Avoir à refaire cette pièce loupé serait vraiment agassif ! **Page 48**

➤ **Concrétiser le boîtier qui enfermera l'ensemble.**

Lorsque les cinq faces qui constituent cet élément principal du coffret sont terminés, on les soude les unes sur les autres par utilisation du diluant cellulosique en veillant à la parfaite géométrie. Les plaques sont placées parfaitement en alignement des bords et on veille aux diverses perpendicularités. Durant ces opérations il faut prendre garde à ne pas souiller les faces extérieures avec du diluant ou avec nos empreintes digitales. Donc ne pas mouiller exagérément le pinceau qui sert à répartir le diluant sur les surfaces à souder. Il est en fait très délicat de réussir parfaitement ces opérations pour un débutant, et si des coulures malencontreuses gâchent la beauté du boîtier, ce n'est pas tragique. Vous les poncez avec du papier abrasif très fin et vous pouvez peindre ce matériau. Utilisez alors une "bombe aérosol". Ce n'est pas très écologique, mais le résultat sera parfait. Surtout bien respecter les repères d'[Image 45.JPG](#) ou vous vous tromperez inexorablement. Du reste, regardez bien la Fig.145 avec en **A** la prévision informatique et en **B** le résultat final. Trouvez l'erreur !

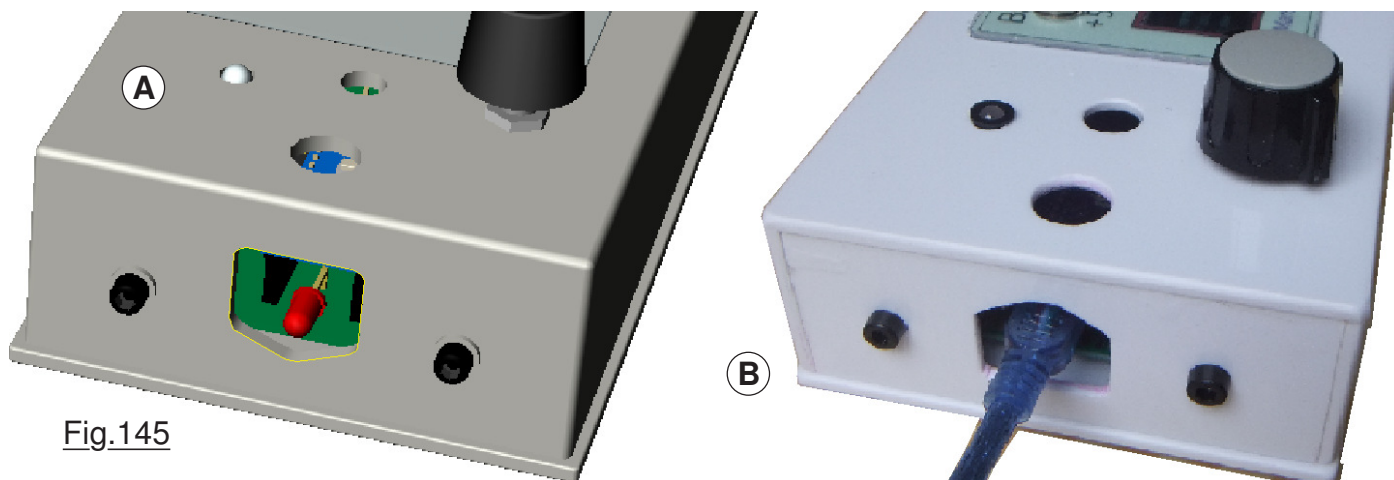


Fig.145

- Ben oui, comme avant de "souder" les plaques je les passe entièrement à l'alcool ménager pour qu'elles soient parfaitement propres, les repères sont alors effacés. Et je me suis trompé ! Trop tard pour rectifier, car je m'en suis aperçu quand la machine était terminée. Comme ça ne gêne pas et que l'ouverture est masquée par le bouchon ... vogue la galère !

Assembler le boîtier consiste coller les cinq faces, puis à renforcer les angles ce que l'on voit sur [Image 53.JPG](#) et [Image 54.JPG](#) en observant sur la première le trou dans lequel se logera l'ergot anti rotation du codeur rotatif. Notons au passage que ce trou ne traverse pas, il présente une profondeur suffisante d'environ 2mm. Si d'infortune vous traversez en perçant, ce n'est pas grave du tout car de toute façon il serait caché par la rondelle frein et le gros bouton. Puis, on réalise les deux longerons constituée chacun de trois bandes de 12mm de largeur. C'est dans la bande centrale que sont logés les écrous prisonniers. (Trois par longeron.) On soude alors ces deux longerons sur la base de la face coté opérateur et coté opposé. Quand l'ensemble est bien rigidifié, on place le boîtier sur la semelle et on repère sur cette dernière avec précision la position des six trous de passage des vis d'assemblage. Cette opération est très délicate et il est probable que ces trous ne seront pas parfaitement alignés sur les écrous inclus dans les longerons. Aussi, dans un premier temps on perce à exactement 3mm. Puis on place la semelle bien centrée sur les boutons du clavier. On repère alors la position exacte des trous de passage

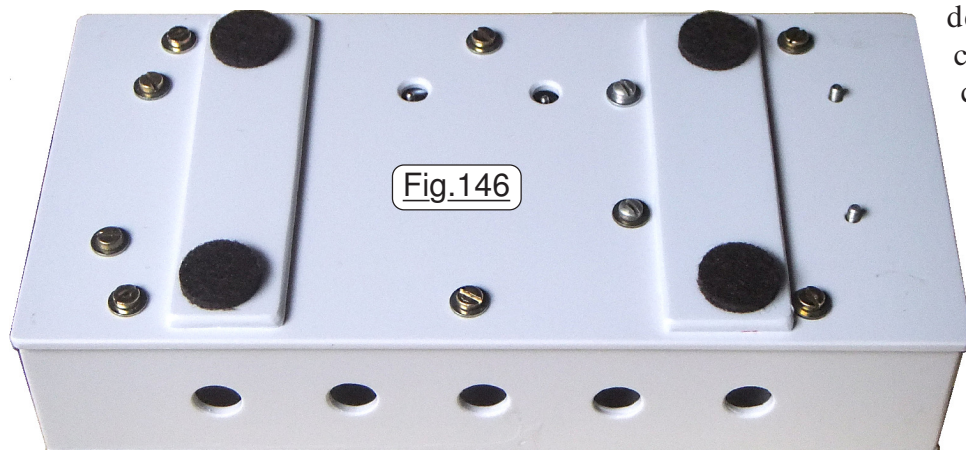


Fig.146

des vis et avec une petite lime cylindrique on les agrandi en les centrant. Vous pouvez remarquer qu'effectivement sur la photographie [Image 50.JPG](#) que ces orifices sont un peu plus grands que les autres.

Enfin, lorsque l'on peut assembler la semelle parfaitement centrée sur la LED et les touches du petit

clavier sur le boîtier, comme le montre [Image 50.JPG](#) on trace les contours précis. On aboutit alors au résultat parfait d'[Image 51.JPG](#) sur laquelle la semelle dépasse tout le tour d'environ 1mm. Enfin, pour terminer entièrement la semelle, on colle sur le dessous quatre patins en feutres. Toutefois, il faut comme présenté sur la Fig.146 les surélever de 3mm en soudant deux traverses de polystyrène choc, car leur épaisseur est insuffisante et la visserie dépassant sur le dessous talonnerait sur le bureau. Pour achever les travaux sur le boîtier, on notera sur [Image 52.JPG](#) l'usage d'une traversée de centrage pour la LED et la gravure des polarités à respecter sur le côté d'insertion de la pile rechargeable.

18) Câblage des éléments du coffret.

C'est dans le dossier <Câblage final> de la <Galerie d'images> que sont préservées les photographies qui illustrent ce chapitre.

Avant de procéder au câblage, on commence par réaliser la petite vitre d'[Image 55.JPG](#) qui est taillée dans le couvercle d'un boîtier qui contenait un Compact Disc. Avec l'avènement des clefs USB, CD et DVD qui servaient de mémoire de masse sont définitivement mis au rebut. Avant de les éliminer à la décharge, je récupère les couvercles translucides de leurs boîtiers qui sont parfaits pour faire des petites vitres faciles à usiner et à percer. La Fig.147 confirme que la transparence de ce matériau est parfaite. On imprime une petite étiquette avec l'ordinateur et l'esthétique est garantie. Quand cet élément est disponible et l'étiquette imprimée, on les assemble sur la face du dessus du boîtier. Ils sont immobilisés par deux petits boulons ϕ M2 et par les deux inverseurs.

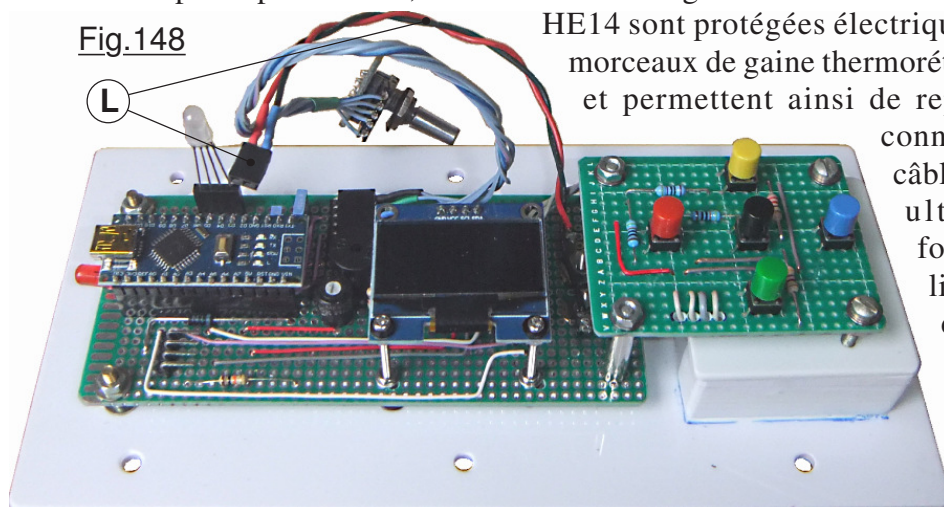
ATTENTION : Le moindre que vous allez serrer un peu trop fort ces divers éléments, et c'est la fissure assurée sur la belle plaque transparente. Aussi il est impératif de serrer très modérément ces divers éléments. Du coup ils risquent sur le long terme de se désolidariser. C'est la raison pour laquelle sur [Image 56.JPG](#) on observe qu'ils sont collés avec du vernis à ongles. Même punition sur [Image 57.JPG](#) pour les deux douilles pour fiches bananes.

Durant le soudage des inverseurs, vous pouvez être certains que des petites gouttes de résine vont sauter et tomber sur la vitre qui serait alors vulnérable lors de son nettoyage. Elle est parfaitement propre car bien nettoyée à l'alcool domestique avant de se voir immobilisée. Aussi, il est facile de la protéger avant soudure par le morceau de papier de l'[Image 58.JPG](#) immobilisé provisoirement par du ruban adhésif. Il est absolument impératif de pouvoir comme sur la Fig.148 désolidariser totalement la semelle est ses circuits imprimés du coffret pour des raisons évidentes de maintenance. Outre le connecteur HE14 qui relie le codeur rotatif au circuit imprimé principal, on observera en **L** la ligne qui relie les deux bornes de la pile vers les inverseurs. La Fig.149 précise le schéma de câblage ainsi que les connecteurs utilisés. Les fils de câblage gris sont issus de nappes plates de connectiques d'ordinateurs. En séparant les fils et en les torsadant on obtient des lignes "compactes" et souples. Compte tenu des faibles courants consommés par la petite unité, ces fils sont très largement dimensionnés. Les soudures sur les picots des

Fig.147



Fig.148



HE14 sont protégées électriquement et mécaniquement par des morceaux de gaine thermorétractable. Ces derniers sont colorés et permettent ainsi de repérer aisément l'orientation des connecteurs. Sur [Image 59.JPG](#) le câblage est terminé et l'on procède aux ultimes vérifications de bon fonctionnement. Quand on réalise la

ligne qui va du clavier vers le HE14 qui se branche sur le codeur rotatif, bien faire attention au fait que les fils ne vont pas broche à broche mais sont croisés. Si par erreur vous avez interverti **A** et **B**, il sera facile de modifier le

programme Arduino en permutant **3** et **2** dans les définitions en tête du programme **P16**. Enfin, sur la

```
#define Codeur_A 3 // D2 : DT
#define Codeur_B 2 // D3 : CLK
```

photographie d'[Image 60.JPG](#) le coffret est entièrement achevé et notre petite machine pleinement fonctionnelle.

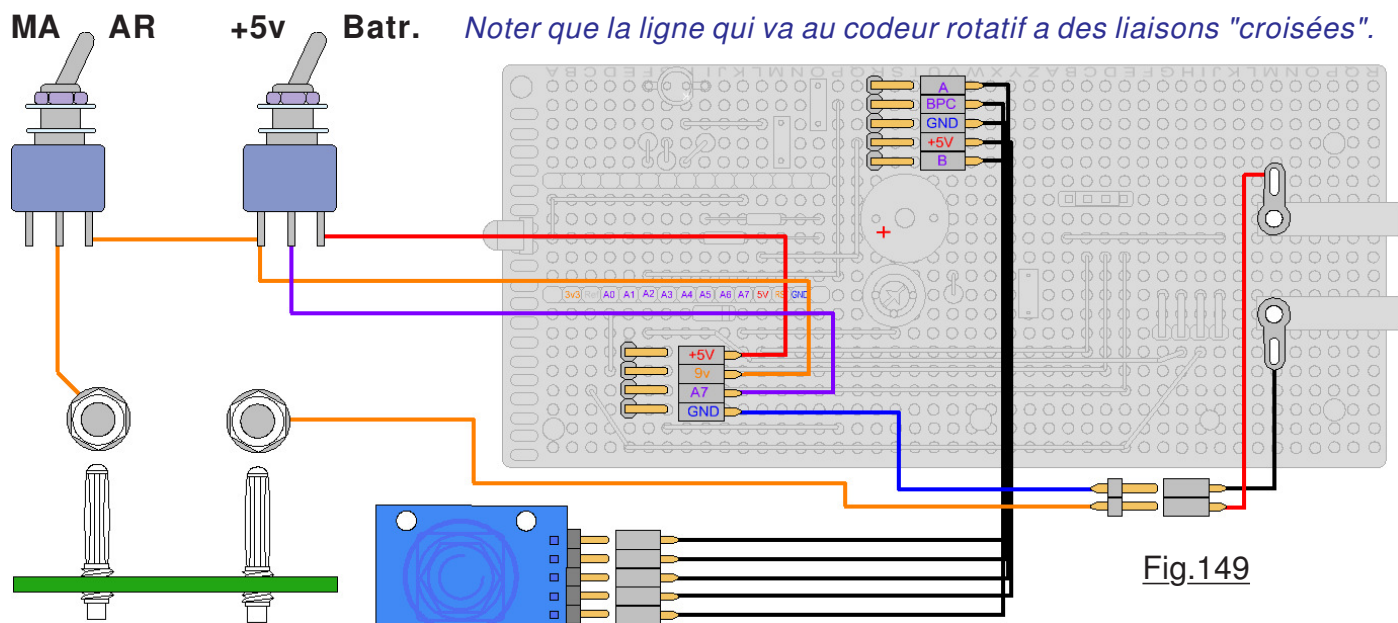


Fig.149

➤ Les derniers petits détails de réalisation.

Cette fois c'est dans le dossier <DIVERS> que sont illustré les propos de ce chapitre. Commençons par la réalisation du sectionneur qui masque la prise mini-USB et autorise l'utilisation de la batterie. Cet élément est réalisé au moyen d'une petite plaque de circuit imprimé simple face **3** dans laquelle on a percé avec l'entraxe précis les deux trous de passage des fiches bananes **1**. Le diamètre de ces trous doit être le plus petit possible compatible avec le passage du corps fileté de **1**. Quand on place le total sur les deux douilles banane $\phi 2\text{mm}$ les deux broches doivent être parfaitement alignées. On commence par les souder en un point coté cuivre, c'est à dire coté caché sur la Fig.150 issue des études sur l'ordinateur. Les deux fiches étant immobilisées, on les insère dans les douilles du coffret pour en parfaite l'orientation. Puis on les soude entièrement comme sur [Image 61.JPG](#) sur laquelle on voit également le petit capot qui englobera la partie extérieure de cet accessoire. Ce petit boîtier est moulé en PLA sur une imprimante 3D. Vous trouverez dans le dossier dédié les deux fichiers nommés **Capot sectionneur** pour éventuellement en faire un clone. Puis comme on le voit sur [Image 62.JPG](#) on prépare le fil de liaison **2** que l'on soude sur les deux fiches bananes visible en [Image 63.JPG](#). Rien à voir avec une quelconque augmentation de la section cuivrée de passage du courant, l'unité électronique consommant moins de 30mA. Cette "grosse barre" cuivrée sert à procurer aux deux fiches une grande rigidité mécanique. Ensuite, sur [Image 64.JPG](#) on colle une petite bague qui crée une zone dans laquelle la colle Araldite ne passera pas. On place une bonne couche de cette colle bi-composants dans le petit moule translucide et on y place le circuit imprimé de telle sorte qu'il affleure en [Image 65.JPG](#) le rebord périphérique. Colle bien durcie on vernit le cuivre pour ne pas qu'il perde sa belle couleur par oxydation et sur [Image 66.JPG](#) le "bouchon sectionneur" est terminé. La bague noire sert à créer un canal de passage de la lumière émise par la LED rouge que l'on peut ainsi observer par transparence.

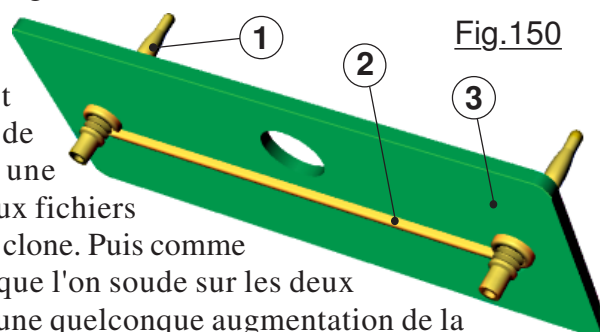


Fig.150

Coté logement de la pile d'alimentation la plaquette d'obturation a été conçue pour tourner autour de la vis située à droite sur [Image 67.JPG](#) avec visible les gravures de polarité. Si vous inversez le sens, une diode protège la carte Arduino NANO. Sur [Image 68.JPG](#) le logement est entièrement dégagé. On replace la pile 9v qui va contre les deux lamelles souples de la Fig.137 sachant que la longueur du logement est volontairement un peu trop importante. Ainsi, comme montré sur la Fig.151 on doit **intercaler une cale** dont l'épaisseur sera choisie pour **obtenir un contact franc sans pour autant faire fléchir exagérément les deux lamelles de contact**. Ici s'achève la description de l'unité autonome. Je vous souhaite de trouver dans ces pages la motivation de vous créer une telle petite Machine de Turing qui nous réservera d'agréables soirées d'hiver bien au chaud à lui soumettre vos algorithmes ...

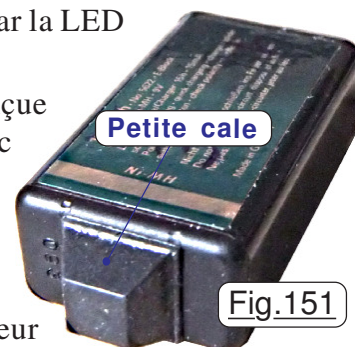


Fig.151

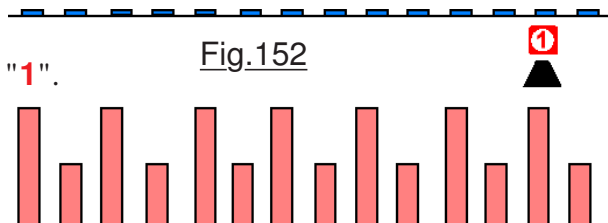
19) Une idée d'un programme simple à développer pour les débutants.

Restant persuadé que celles et ceux qui vont créer leur propre version de Machine de Turing, vous allez inexorablement commencer par vous faire plaisir et soumettre à cette dernière une palanquée de programmes et en particuliers ceux qui sont proposés dans le dossier **<Fiches de PROGRAMMES>**. Toutefois, après avoir assouvi cette boulimie d'expérimentations, viendra forcément le moment où vous désirerez voler de vos propres ailes. Aussi, dans ce chapitre je vous propose un petit programme pas très compliqué que vous pourrez soumettre à votre sagacité :

L'idée consiste à remplir entièrement le plateau avec comme schématisé en Fig.152 une alternance de "0" et de "1".

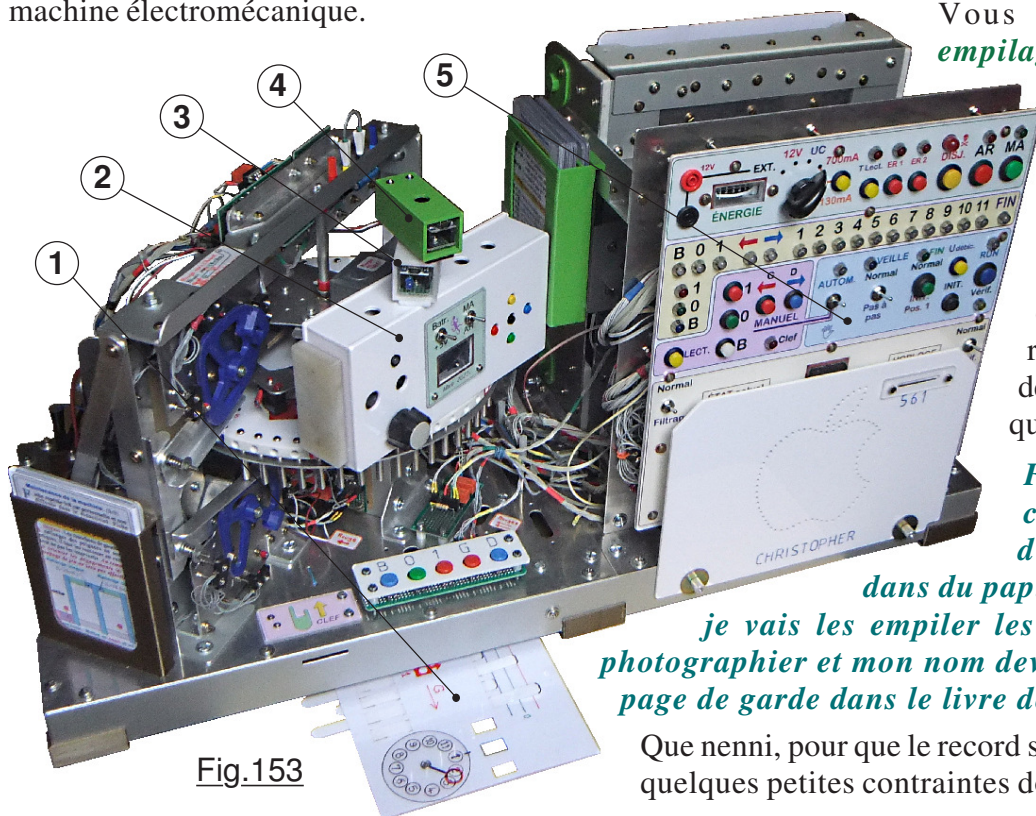
Toutefois il faudra respecter les contraintes suivantes :

- Le déplacement de la tête L/E sera vers la droite.
- Commencer par écrire tous les "1".
- Puis intercaler tous les "0".
- Fin du programme dès que le plateau est saturé.
- Optimiser l'algorithme ... ça va sans dire !
- Le plateau initial sera rempli de "B". L'Origine ainsi que la tête de L/E seront en position n°1.



20) Encore un record mondial à battre !

Observez bien la Fig.153 sur laquelle a été immortalisé ce défi à relever. À première vue l'image montre notre prototype électromagnétique au repos. Si on y porte une attention plus soutenue, on constate qu'en 1 la grosse machine est posée sur le petit exemplaire en papier. En 2 se trouve la machine autonome. Posée sur cette dernière ma petite unité élémentaire 3 moulée dans du PLA translucide. Posée tout en haut en 4 la petite jumelle verte que je n'avais pas encore envoyé à Sylvain. Enfin en 5 la belle machine électromécanique.



Vous admirez en réalité **un empilage** de pas moins de **cinq machines de Turing**, c'est le défi que je vous propose de relever. Et encore, vous avez de la chance car j'en avais une sixième à ajouter sur le dessus, mais lorsque j'ai réalisé cette image je l'avais déjà envoyée à mon ami Alain qui habite en Suisse.

Fastoche vont claironnée certains, je vais en faire découper une trentaine dans du papier par les élèves de CM2, je vais les empiler les unes sur les autres, les photocopier et mon nom deviendra illustre et sera en page de garde dans le livre des records !

Que nenni, pour que le record soit validé il faudra respecter quelques petites contraintes dont voici la liste :

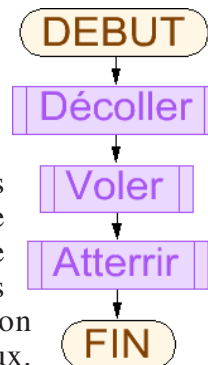
- Vous vous engagez moralement à ne pas fournir une image truquée,
- Une seule machine au maximum sera en carton ou en papier,
- Mécaniques, électromécaniques, en or massif, peu importe leur technologies. Comme ici certaines peuvent être totalement identiques mais ... Toutes doivent fonctionner correctement et se montrer opérationnelle le jour de la prise de vue. (*Ben oui, c'est un minimum.*)
- Toutes devront avoir été entièrement réalisées par le postulant avec interdiction de se faire aider par un quelconque service ou une aide extérieure.

Bon courage les amis, la gloire vous attend ... Page 52

21) Machine du Turing et SUBROUTINE.

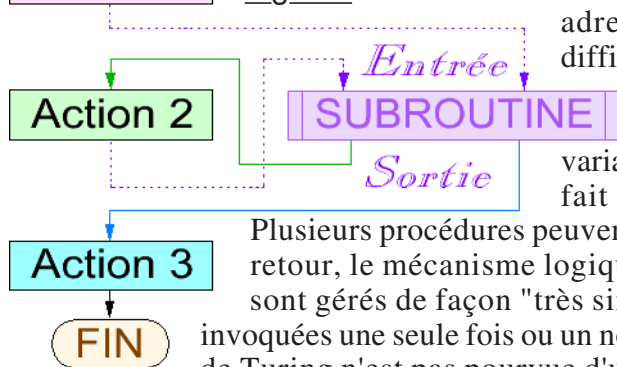
L'intégralité des logiciels actuels utilisent des sous-programmes. On peut concevoir des "subroutines" qui ne sont invoquées qu'une seule fois. Le but consiste à traiter dans des "blocs" faciles à identifier des séquences spécifiques. Par exemple le programme de la Fig.154 a été conçu en trois blocs qui sont invoqués les uns après les autres. Si l'on choisit avec pertinence les identificateurs des procédures, on peut presque deviner ce que fait le programme. Plus généralement, lorsqu'une séquence se retrouve plusieurs fois dans un logiciel, il devient moins coûteux de ne l'écrire qu'une seule fois sous forme d'une *subroutine*, et de l'invoquer chaque fois qu'il y en a besoin. Cette façon d'organiser un logiciel est naturelle, toutefois elle engendre un problème technique épineux.

Fig.154



En effet, dans la séquence de la Fig.154 chaque bloc sait où se trouve la suite du programme, car une seule transition de "branchement" est utilisée. Il en va tout autrement sur la Fig.155 où l'on fait appel deux fois à la *subroutine*, avec chaque fois une "adresse de retour" différente. C'est précisément dans la gestion des

Fig.155



adresses de retour dans le programme appelant que réside la difficulté. Sans entrer dans les détails, sur les processeurs actuels un dispositif logique interne est dédié à la gestion des appels et retours de procédures, ainsi que celle des variables dites locales. Chaque fois que le programme "principal" fait appel à une procédure, il em

PILE l'adresse de retour. Plusieurs procédures peuvent être invoquées. Dès que l'une d'elle trouve l'instruction de retour, le mécanisme logique dé**PILE** la dernière adresse enregistrée. C'est ainsi que sont gérés de façon "très simple" les appels à fonction ou procédures, qu'elles soient invoquées une seule fois ou un nombre quelconque. Il se trouve que par principe une Machine de Turing n'est pas pourvue d'une telle organisation logique. (**PILE** et "stack pointeur".)

Nous sommes donc naturellement incités à nous demander si intégrer l'utilisation d'une procédure invoquée plusieurs fois est possible en respectant scrupuleusement le concept de l'illustre mathématicien.

➤ Un défi à relever.

Pratiquement tous les mathématicien avec pour complices les informaticiens s'accordent à considérer que *tout ce que peut faire un ordinateur actuel quelle que soit sa puissance, une machine de Turing existe et en est capable. La réciproque n'est pas forcément vraie.* Bien entendu il n'est absolument pas question ici de

le prouver ! En revanche, si nous arrivons à coder une organisation logicielle telle que celle de la Fig.155, non seulement on aura ouvert une piste de recherche très intéressante, et surtout on aura fait pencher la balance du bon côté, sans pour autant que ce soit une démonstration. Dans ce chapitre je vous invite à relever le défi. Pour que l'exercice soit démonstratif, on va utiliser la machine la plus puissante mise à notre disposition, c'est à dire que l'on va activer le mode ÉTENDU pour disposer de 20 transitions dans la matrice virtuelle. Je vous propose les contraintes suivantes :

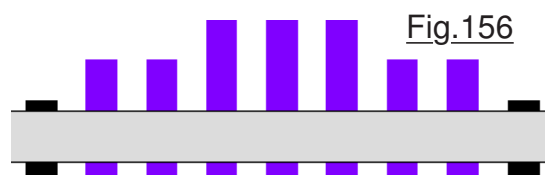


Fig.156

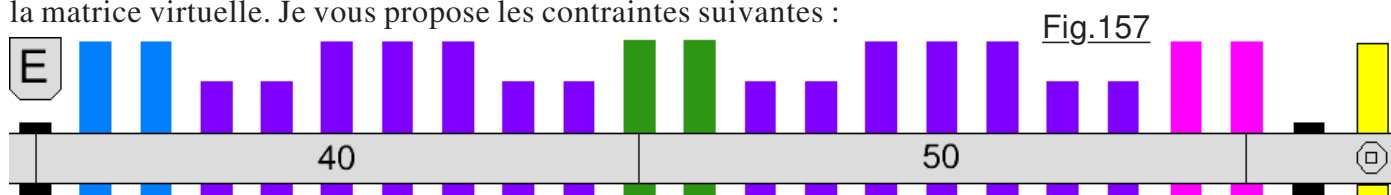
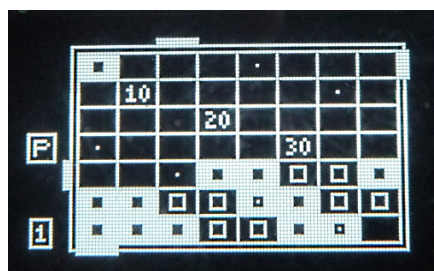


Fig.157

- La structure de l'algorithme sera celle de la Fig.155 et les trois actions séquentielles seront identiques et forceront deux BITS à "1", et effectueront deux rotations vers la Droite.
- Le sous-programme (Subroutine.) générera la suite de la Fig.156 par des rotations également à droite. Du coup le résultat final sera celui de la Fig.157 lorsque "F" sera rencontré.
- Le traitement des adresses de retour de procédure sera forcément géré par une donnée binaire, car c'est la seule entité qui puisse coexister sur le plateau de la machine.
- Le plateau initial sera effacé et ne contiendra que des "B", l'origine ainsi que la tête de Lecture / Ecriture seront en position n°1 du carrousel.

Une solution possible pour relever ce défi est proposée dans le tableau page suivante. Ce dernier est paginé sens dessus dessous pour vous éviter de voir cet algorithme sans le vouloir, ainsi le charme de vous "cogner" à cet algorithme et d'en découvrir la subtilité est préservé. Il faut penser "Turing", c'est à dire traiter le comportement du programme et de la **PILE** comme une donnée binaire ordinaire. **Page 53**



Dans cet algorithme la transition n°1 constitue la donnée d'aiguillage et joue le rôle de la **PILE** de sauvegarde des adresses de retour. Si on lit un "B" on écrit un "0" qui vectorisera le premier retour de sous-routine. Puis en Tr2 on saute le séparateur "B". En Tr3 et Tr4 on traite l'ACTION 1 avec appel de la sous-routine en Tr13. Si on lit un "0" en Tr1 on écrit un "1" pour la deuxième adresse de retour et on branche sur l'ACTION 2 en Tr7, mais il faut avant revenir sur la gauche des données avec Tr5 et TR6. L'ACTION 2 en Tr7 et Tr8 se termine par l'appel de la sous-routine en Tr13. La procédure Tr13 à Tr19 réalise son travail puis en Tr20 branche sur Tr1 qui gère les adresses de retour. En Tr1 on lit un "1", l'algorithme branche sur Tr9 qui avec Tr10 ramène la tête de L/E sur la fin des données. Enfin en Tr11 et Tr12 l'ACTION 3 est exécutée et le programme se termine avec le "F".

62 trous

84 cycles horloge.

Ben Môa môa, je déplore que le monde actuel soit de plus en plus PROCÉDURAL et ce n'est pas l'intelligence artificielle qui va modifier cette triste évolution !

État	Lectr	Ecrit	Mvt	ÉTAT
1	1:		D	T9
	0:	1	D	T5
	B:	0	D	T2
2	1:			
	0:			
	B:		D	T3
3	1:			
	0:			
	B:	1	D	T4
4	1:			
	0:			
	B:	1	D	T13
5	1:			
	0:			
	B:		D	T6
6	1:		D	
	0:		D	
	B:			T7
7	1:			
	0:			
	B:	1	D	T8
8	1:			
	0:			
	B:	1	D	T13
9	1:			
	0:			
	B:		D	T10
10	1:			
	0:		D	
	B:			T11

État	Lectr	Ecrit	Mvt	ÉTAT
11	1:			
	0:			
	B:	1	D	T12
12	1:			
	0:			
	B:	1	D	F
13	1:			
	0:			
	B:	0	D	T14
14	1:			
	0:			
	B:	0	D	T15
15	1:			
	0:			
	B:	1	D	T16
16	1:			
	0:			
	B:	1	D	T17
17	1:			
	0:			
	B:	1	D	T18
18	1:			
	0:			
	B:	0	D	T19
19	1:			
	0:			
	B:	0		T20
20	1:			
	0:			
	B:		G	T1

22) La réalisation des petits manuels.

Faisant partie intégrante de mes didacticiels, généralement des petits livrets accompagnent le tutoriel. Dans ces livrets au format A5 on trouve souvent le manuel d'utilisation du projet, un autre document sur l'agencement du logiciel etc. Par exemple, pour ce projet de Machine de Turing Autonome, le programme utilise un afficheur dont la bibliothèque fournie met à disposition des protocoles très performants. Encore faut-il pouvoir les mettre en œuvre correctement. Pour celles et ceux qui chercheront à appréhender le logiciel qui anime la machine, le petit livret décrivant **U8glib** me semble indispensable. Que ce soit pour un livret informatique ou un manuel d'utilisation du prototype, dans tous les cas il importe de s'y prendre correctement pour imprimer et assembler le petit.

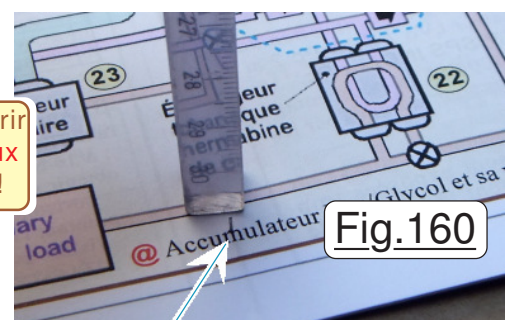
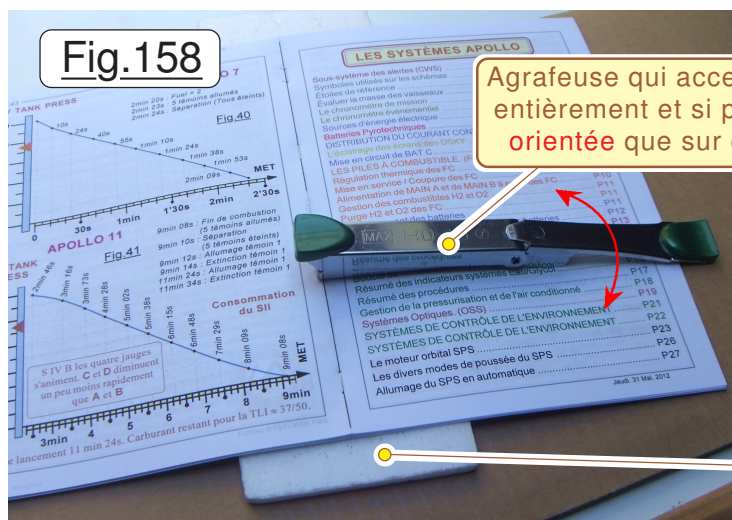
➤ Réalisation matérielle d'un petit livret au format A5.

Agencé à un format A5, les faibles dimensions de ce manuel en font un document parfaitement adapté à son usage. Pas trop petit, les dessins et schémas sont de dimensions suffisantes, pas trop gros, il trouve facilement sa place dans un tiroir de rangement. Les fichiers tels que **Bibliothèque U8glib.pdf**, sont prévus pour être imprimés RECTO/VERSO. Il importe donc de choisir du papier d'épaisseur "normale" pour ne pas que l'encre ne traverse. **Papier recyclé méga écolo** s'abstenir ! Personnellement je commence par imprimer toutes les pages impaires. Puis, paquet de feuilles replacé sur le bac à papier de la machine **CORRECTEMENT ORIENTÉ ET DANS LE BON ORDRE** je fais imprimer toutes les pages paires. Pour cette phase il me semble moins risqué d'opérer page par page, et vérifier à chaque tentative que deux A4 n'ont pas été "aspirés" par le mécanisme qui tracte les feuilles sous les têtes d'impression. Vous ne perdrez ainsi qu'une seule épreuve, alors que si vous engagez l'opération pour toutes les feuilles ... c'est tout le paquet qu'il faudra entièrement réimprimer. Bien réfléchir quand on replace le paquet dans le bac de la machine, car les pièges sont nombreux. (*Inverser le haut et le bas, face sur le dessus qui n'est pas la bonne, pages entassées dans l'ordre incorrect ...*)

Éventuellement tester avec une feuille brouillon en mode économique noir et blanc. Puis, quand tout est imprimé, réaliser l'assemblage est relativement élémentaire :

- 1) Commencez par plier toutes les pages en deux. (*Et si possible du bon côté !*)
- 2) Trouvez une plaque de polystyrène ou du carton bien classique. (*Voir Fig.158*)
- 3) Positionnez les pages bien à plat et surtout bien les unes cadrées sur les autres.
- 4) Avec une petite agrafeuse qui accepte de s'ouvrir complètement mettre en place quatre "crochets".

ATTENTION : Quand on appuie sur l'agrafeuse il faut bien la tenir latéralement car elle veut se décaler sur les cotés. Du coup comme montré en Fig. 159 l'agrafe est mal enfoncée et se plie. Quand une agrafe s'est tordue, la retirer avec un cutter et en placer une deuxième exactement au même endroit. Le deuxième essai sera le bon ...



- 5) Retourner le livret **sur un support rigide** et fermer les agrafes à la main avec un outil quelconque. Dans mon cas j'utilise une règle de section carrée comme montré sur la Fig. 160 sur laquelle à peine visible on voit un coté de l'agrafe non encore replié.

Notez que pour faciliter la tâche les pages sont numérotées verticalement au centre pour repérer plus facilement l'ordre d'assemblage. Gutembérisez bien les ami(e)s ...