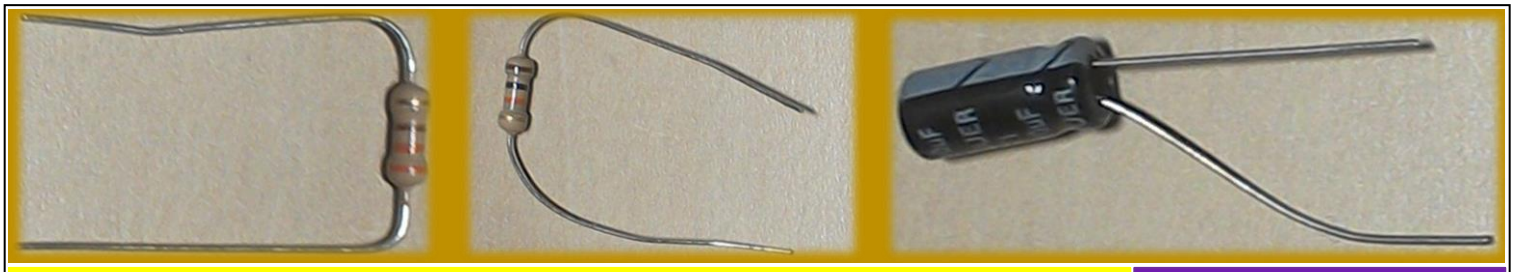
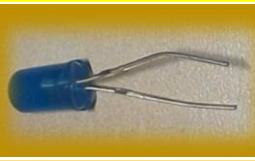
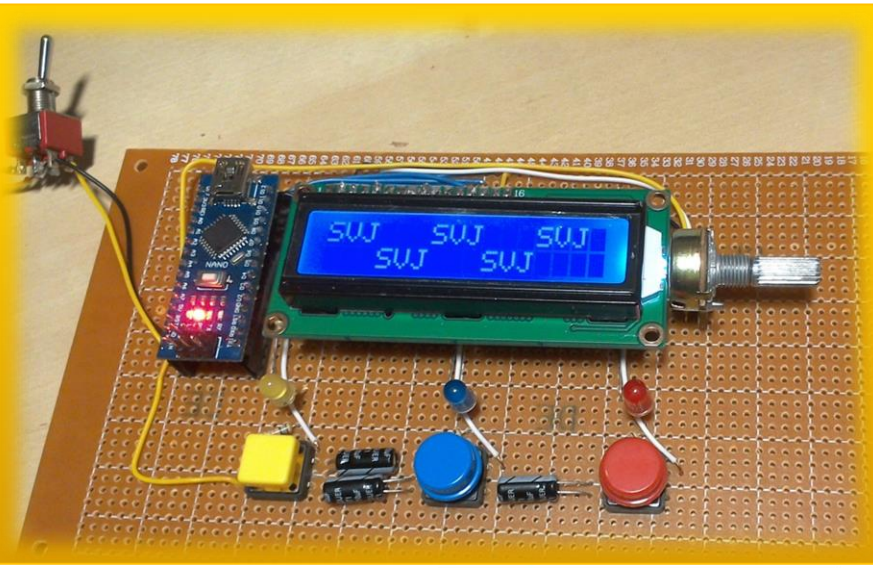


Morscipio

Boitier Décodeur de Morse



2016



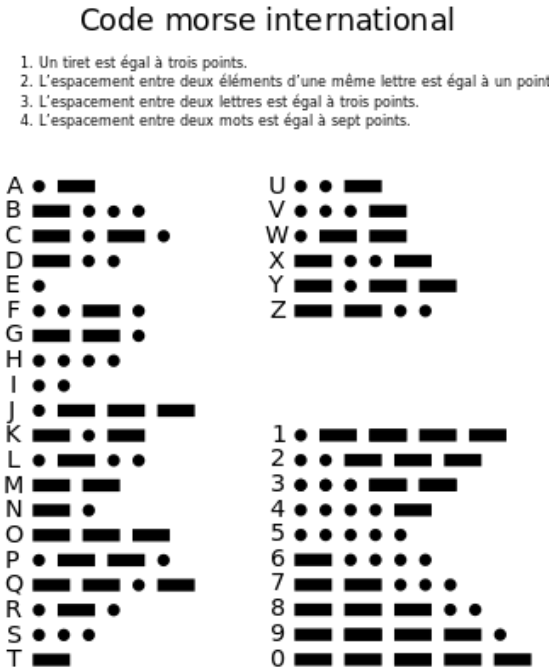
Morscipio

Contenu

Le Besoin Initial

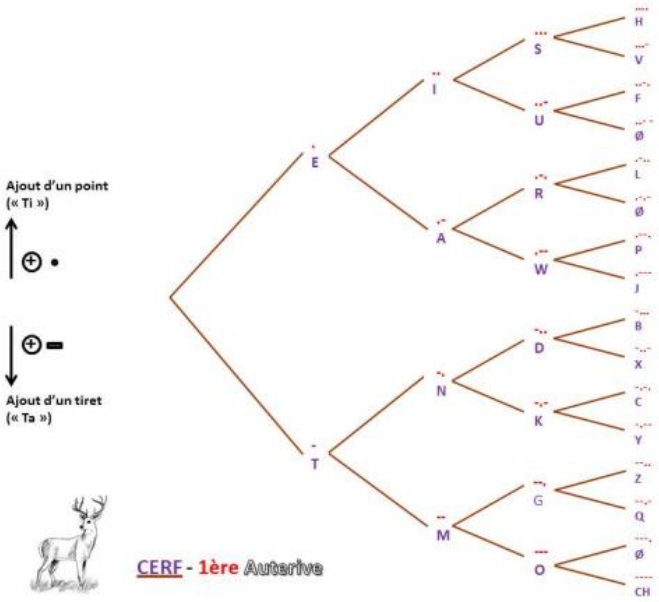
Le Problème

Je suis scout d'Europe, et lors de nos sorties, des informations nous sont souvent transmises en morse. Ainsi, il est possible que pour le début d'un jeu, on reçoive un message en morse nous disant où se trouve le prochain rendez-vous. Le morse est un code international qui à chaque lettre fait correspondre une série de signaux courts et/ou longs. C'est ainsi que se faisaient et se font encore beaucoup de communications. On peut utiliser ce code de différentes manières : avec des signaux sonores, lumineux…



Correspondance Alphabet-Morse

J'ai besoin, lors des sorties, de pouvoir décoder des messages facilement. Je voulais donc fabriquer quelque chose qui permettrait de recevoir des messages morses, et de les décoder le plus rapidement possible. Tout d'abord, je me suis rendu compte que lorsque l'on émet un message, la plaquette ci-dessus était adaptée, mais lorsque l'on veut recevoir un message, elle ne l'est plus du tout. J'ai donc fabriqué une autre plaque avec les correspondances lettres signaux, mais en forme d'arbre (un peu comme les arbres de probabilités en maths):



Correspondance Morse-Alphabet

Ainsi, il était plus facile de retrouver la lettre qui correspond à un signal donné. Cependant, je n'étais pas satisfait de cette nouvelle méthode : les papiers s'envolent, se déchirent, s'abiment… ou servent pour allumer le feu ! Je voulais donc quelque chose d'encore plus simple à utiliser, et dont la traduction ne demande pas d'effort supplémentaire, ainsi, je voulais réduire le temps de réception des messages au maximum. Comme je débutais en électronique, j'ai décidé de me lancer dans un projet électronique : j'ai progressé techniquement, et le produit final a une réelle utilité.

Le Cahier des Charges

La description fonctionnelle de ce que je voulais fabriquer était donc simple :

Fonction principale : Décoder un message morse

Priorités :

- Simplicité d'utilisation
- Facilité de transport :
  - A peu près étanche
  - Pas trop lourd ni encombrant
  - Assez robuste
  - Fonctionnement sur pile(s)/batterie

Budget : 15-20€ (environ)

Avec toutes ces contraintes, j'avais besoin de l'électronique pour résoudre le problème. Comme je débutais Arduino, cela me faisait un bon projet à réaliser.

La Conception

Les Pièces Electroniques (Composants)

Les Besoins

Pour faire ce décodeur, il y a deux types de pièces électroniques : celles qui interagissent avec l'utilisateur, et celle qui n'interagissent pas avec celui-ci. J'ai d'abord fait l'inventaire de ce que l'utilisateur aurait à utiliser :

- **3 Boutons** : Pour les signaux **courts**, **longs**, et les **espaces** entre les lettres
  - ⇨ Ou boutons **point** ; **tiret** et **slash**
  - **3 LEDs** témoins pour les 3 boutons
- **1 Ecran LCD** pour afficher le texte décodé

Les Nouvelles Règles du Code Morse

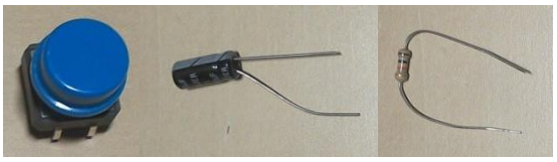
Ainsi, le morse se code facilement suivant ces règles :

1. Les tirets et les ponts se font avec les boutons « Tiret » et « Point »
   - ⇨ Equivalent à 3 points pour un tiret
2. Un espace entre deux éléments d'une même lettre se fait automatiquement
   - ⇨ Equivalent à 1 point d'attente
3. Un espace entre deux lettres, se fait par 1 slash (bouton « Slash »)
   - ⇨ Equivalent à 3 points d'attente
4. Un espace entre deux mots, se fait par 2 slashs (bouton « Slash »)
   - ⇨ Equivalent à 7 points d'attente

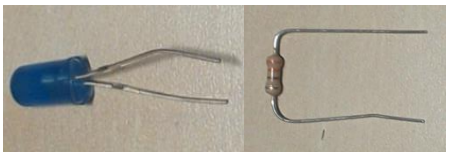
La Liste Complète des Composants

A partir de là, il est facile de déterminer tous les composants nécessaires :

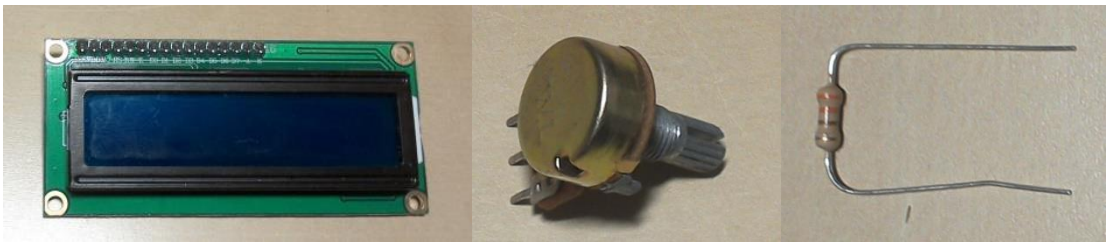
- 3 Boutons :
  - 3 Capacités (10µF)
  - 3 Résistances (10kΩ)



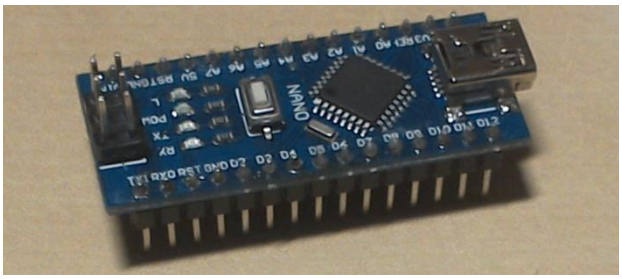
- 3 LEDs :
  - 3 Résistances (330Ω)



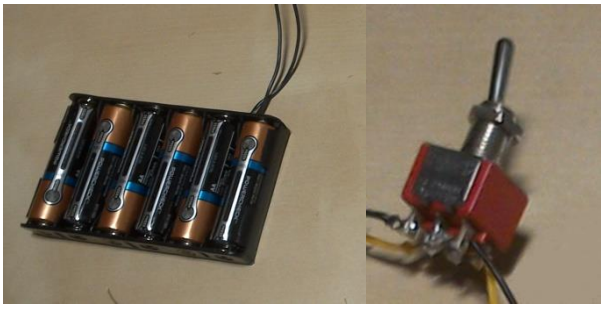
- 1 Ecran LCD :
  - 1 Potentiomètre (10kΩ) pour le contraste de l'écran
  - 1 Résistance (330Ω) pour le rétro-éclairage

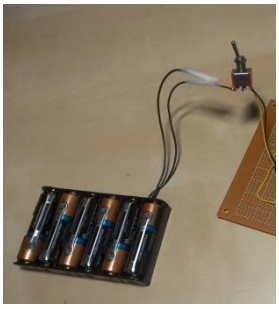


- 1 Carte Arduino



- 1 Boitier pour les piles :
  - 1 Interrupteur ON/OFF pour économiser les piles

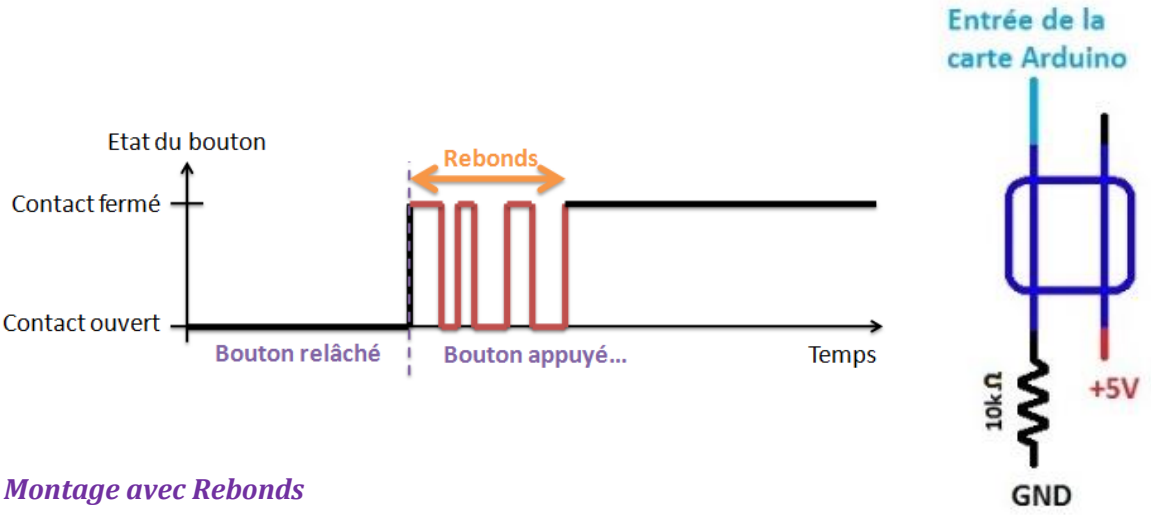




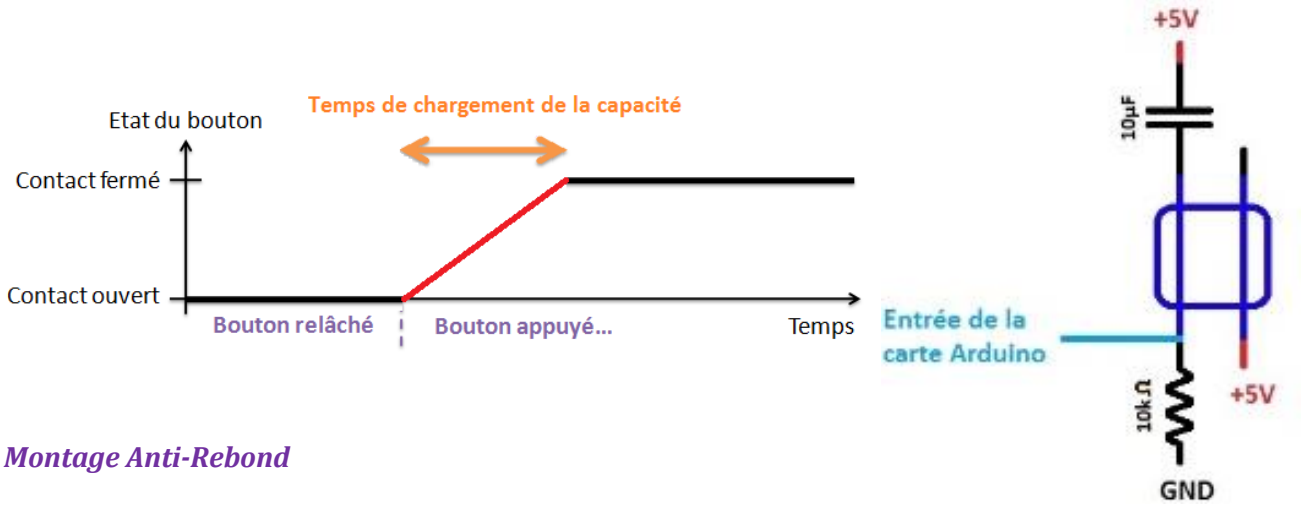
Les résistances des boutons sont nécessaires à leur fonctionnement.

Les capacités servent à limiter l'effet rebond des boutons : lorsque l'on appuie sur un bouton, celui-ci vibre, et le microcontrôleur (ici, la carte Arduino) "croit" que l'on a appuyé plusieurs fois. Pour ne pas rencontrer ce problème, on utilise souvent des capacités, qui mettent du temps à se charger, ainsi, on arrive à lisser le signal, et la carte a bien une seule impulsion de bouton.

Schémas des montages :



Montage avec Rebonds



Montage Anti-Rebond

Les résistances des LEDs servent à limiter la tension pour ne pas les faire griller.

Le potentiomètre permet de régler le contraste de l'écran LCD.

La carte Arduino va permettre la communication des instructions aux différents composants grâce à des entrées et un programme écrit préalablement.

Le boitier de piles permet la portabilité du circuit, qui n'est plus dépendant en énergie de l'ordinateur.

Le bouton ON/OFF permet de ne pas laisser le circuit branché sur les piles en permanence, et ainsi de faire des économies considérables de piles.

Le Programme

Le programme est ce qui permet à la carte Arduino de "savoir" ce qu'elle doit faire. En effet, s'il n'y en a aucun, elle ne fait rien. Ecrit en C++, le programme permet de gérer les entrées et sorties connectés à la carte.

Le programme est composé de deux parties principales, une première, qui réceptionne et stocke les combinaisons de boutons, et une autre qui, à partir d'une de ces combinaisons, détermine la lettre correspondante.

La première partie du code doit gérer les entrées/sorties.

1) Déclarations des Variables et Importation

Avant tout, il faut définir les variables globales du code, c'est-à-dire les broches sur lesquelles sont branchés les différents composants, ainsi que quelques variables nécessaires au code lui-même. Il faut aussi importer les modules complémentaires (un permettant de gérer un écran LCD, nommé <LiquidCrystal.h>, et un permettant de créer des piles de type « First In First Out » nommé <QueueList.h>).

2) Initialisation

Après, il faut définir les entrées et sorties (pins de boutons/LEDs) et démarrer l'écran LCD. On met aussi la liaison série en marche : même si celle-ci n'a d'utilité que si la carte est reliée à un ordinateur, cela sera utile si un bug survient.

3) Gestion des Boutons et des Mises à Jours de l'Affichage

La réception des combinaisons de boutons constitue le corps principal de l'algorithme. Comme la carte ne peut exécuter qu'une seule action à la fois, contrairement à certains langages disponibles sur ordinateur comme Python, on ne peut ici créer qu'un seul thread d'exécution. C'est pourquoi on utilise une boucle (« loop() ») qui se ré-exécute en permanence.

Dans cette boucle la carte teste si l'on appuie sur le 1er bouton, puis le 2eme, puis le 3eme. Si le bouton correspondant au point est fermé, la carte enregistre un point (un 1) dans la pile, si celui du tiret est fermé, elle enregistre un tiret (un 3). La mise à jour de l'affichage se fait lorsque l'on appuie sur le bouton de fin de lettre (le « slash »). Ces trois étapes s'exécutent perpétuellement dans la boucle infinie « loop() ».

La pile du type « First In First Out » (ou « FIFO ») permet d'enregistrer des éléments (ici des 1 pour les points et des 3 pour les tirets), et de les restituer dans le même ordre. Ainsi, lors de la mise à jour, on "dépile", et on obtient le code entré par l'utilisateur, dans le même ordre, ce qui est essentiel pour décoder le message. Le message encore codé est alors envoyé à une fonction qui se charge de déterminer la lettre correspondante.

```
int buts=9;//bouton slash
int butt=11;//bouton tiret
int butp=13;//bouton point

int leds=8;//LED du bouton slash
int ledt=10;//LED du bouton tiret
int ledp=12;//LED du bouton point

#include <LiquidCrystal.h>
//Permet de gérer un écran LCD
#include <QueueList.h>
//Permet de créer une pile où seront stockées les valeurs des signaux

QueueList <int> msg;
LiquidCrystal lcd(3, 2, 4, 5, 6, 7);

int nb=0;//nombre de signaux dans la liste
int colonne=0;//colonne de l'écran LCD sur laquelle est le pointeur
char* aff[17];//variable de stokage pour le défilement du texte
```

1)

```
void setup() {
  lcd.begin(16, 2);
  delay(10);
  //démarrage de l'écran LCD
  Serial.begin(9600);
  delay(10);
  //démarrage de la liaison série s'il y en a une
  pinMode(leds, OUTPUT);
  pinMode(ledt, OUTPUT);
  pinMode(ledp, OUTPUT);

  pinMode(buts, INPUT);
  pinMode(butt, INPUT);
  pinMode(butp, INPUT);
}
```

2)

```
void loop(){
  if (digitalRead(buts)==1){
    digitalWrite(leds, 1);
    if(nb!=0){
      int letter[]={0,0,0,0};
      for(int i=0; i<nb; i++){
        letter[i]=msg.pop();}
      decodeLetter(letter);}
    else{
      afficher(" ");//=espace}
    nb=0;
    while (digitalRead(buts)==1){}//attente du relâchement du bouton
    digitalWrite(leds, 0);
  }
  if (digitalRead(butt)==1){
    digitalWrite(ledt, 1);
    msg.push(3);
    nb++;
    while (digitalRead(butt)==1){}//attente du relâchement du bouton
    digitalWrite(ledt, 0);
  }
  if (digitalRead(butp)==1){
    digitalWrite(ledp, 1);
    msg.push(1);
    nb++;
    while (digitalRead(butp)==1){}//attente du relâchement du bouton
    digitalWrite(ledp, 0);
  }
  delay(5);//attente de 5 ms avant le retour au début de la boucle
}
```

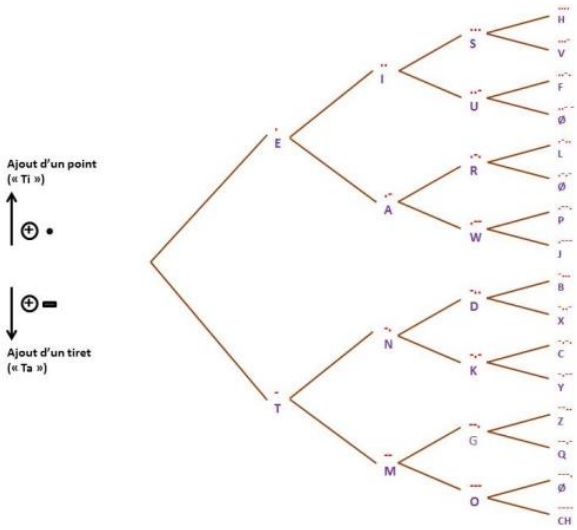
3)

4) Décodage des Combinaisons

La deuxième partie du code doit décoder, à partir des combinaisons de boutons, la lettre morse correspondante.

Pour cela, j'ai réinvesti la plaque déchiffrante que j'avais déjà faite : il suffit de remplacer les branches par des blocs « if (signal n° ==){…} ».

Ainsi, on peut, à l'aide d'une série de conditions successives, déterminer la lettre qui correspond à celle entrée par l'utilisateur.

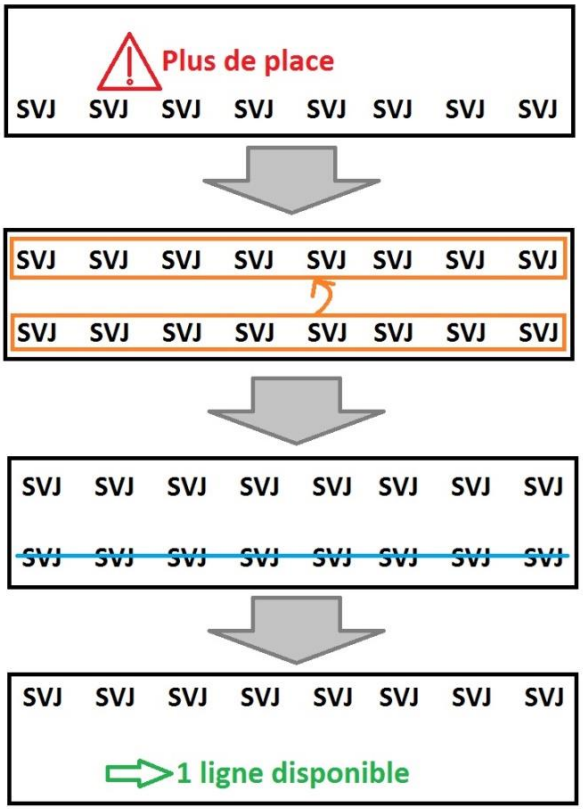


5) Affichage du Texte Décodé

Il reste encore à afficher cette lettre. Grâce au module LiquidCristal.h, cela est très simple, car il existe une fonction « setCursor » et « print ».

Pour une relecture plus simple, j'ai opté pour un défilement ligne par ligne : lorsque la ligne du bas est pleine, celle-ci est recopiée sur la ligne du haut, et la ligne du bas est effacée, libérant ainsi de la place. On recommence l'opération à chaque fois que la ligne du bas est pleine.

On peut le représenter comme ci-après :



Etapes du changement de ligne sur l'écran LCD

```
void decodeLetter(int* letter){
  if (letter[0]==1){
    if (letter[1]==1){
      if (letter[2]==1){
        if (letter[3]==1){afficher("H");}//....
        else if (letter[3]==3){afficher("V");}//...-
        else {afficher("S");}}//...
      else if (letter[2]==3){
        if (letter[3]==1){afficher("F");}//..-.
        else if (letter[3]==3){afficher("?");}//..--
        else {afficher("U");}}//..-
      else {afficher("I");}}//..

    else if (letter[1]==3){
      if (letter[2]==1){
        if (letter[3]==1){afficher("L");}//.-..
        else if (letter[3]==3){afficher("?");}//.-.-
        else {afficher("R");}}//.-.
      else if (letter[2]==3){
        if (letter[3]==1){afficher("P");}//.--.
        else if (letter[3]==3){afficher("J");}//.---
        else {afficher("W");}}//.--
      else {afficher("A");}}//.-

    else {afficher("E");}}//.


  else if (letter[0]==3){
    if (letter[1]==1){
      if (letter[2]==1){
        if (letter[3]==1){afficher("B");}//-...
        else if (letter[3]==3){afficher("X");}//-..-
        else {afficher("D");}}//-..
      else if (letter[2]==3){
        if (letter[3]==1){afficher("C");}//-.-.
        else if (letter[3]==3){afficher("Y");}//-.--
        else {afficher("K");}}//-.-
      else {afficher("N");}}//-.

    else if (letter[1]==3){
      if (letter[2]==1){
        if (letter[3]==1){afficher("Z");}//--..
        else if (letter[3]==3){afficher("Q");}//--.-
        else {afficher("G");}}//--.
      else if (letter[2]==3){
        if (letter[3]==1){afficher("?");}//---.
        else if (letter[3]==3){afficher("?");}//----
        else {afficher("O");}}//---
      else {afficher("M");}}//--

    else {afficher("T");}}//-

}
```

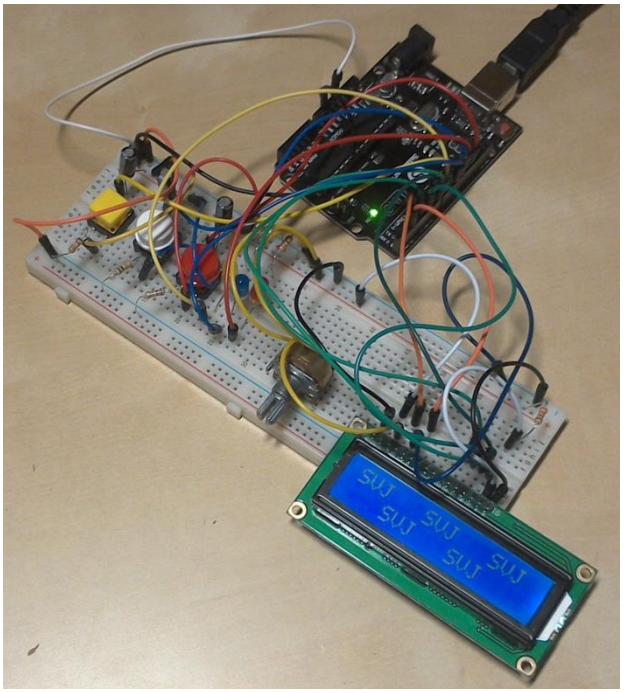
4)

```
void afficher(char* texte){
  Serial.println(texte);
  lcd.setCursor(colonne, 1);
  lcd.print(texte);
  aff[colonne]=texte;
  colonne+=1;
  if(colonne==16){
    lcd.setCursor(0, 0);
    for(int i=0; i<17; i++){
      lcd.print(aff[i]);
      Serial.print(aff[i]);}
    lcd.setCursor(0, 1);
    lcd.print("                ");
    Serial.println("                ");
    colonne=0;
    }
}
```

5)

Le Premier Câblage

Pour le premier essai de câblage du montage, j'ai choisi d'utiliser une carte Arduino UNO, qui a déjà des emplacements pour les fils de connections. Après quelques essais, le montage marchait.



Budget Final

Avant de me lancer dans la réalisation complète (sur plaque d'essai, avec soudures à l'étain), j'ai fait une prévision des dépenses, pour voir si je ne dépassais pas le budget initial.

En utilisant un maximum de récupération (fils et interrupteurs), et grâce au club informatique Info@Lèze, qui met une imprimante 3D à la disposition des membres, j'ai pu réduire grandement les coûts du Morscipio.

De plus, comme les composants électroniques (résistances, LEDs…) de base ne sont pas chers (proche de 0), le coût final du boitier n'est que de 16,77€, ce qui correspond au budget visé au début du projet.

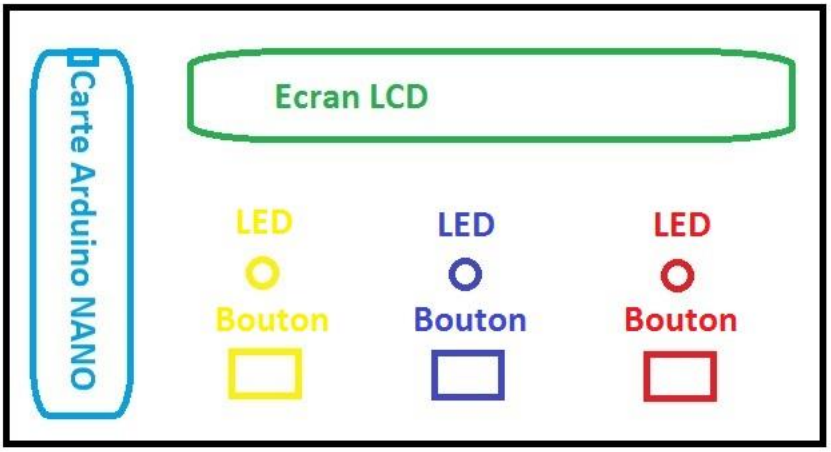
Budget du Morscipio:	
Composants:	**Prix unitaire:**
Carte Arduino NANO	3,00 €
Ecran LCD	2,50 €
Boiter de pile	3,00 €
Interrupteur ON/OFF	Récupéré
Boutons (X3)	0,50 €
Résistances 330Ω (X4)	0,00 €
Résistances 10kΩ (X3)	0,00 €
Capacités 10µF (X3)	0,09 €
LEDs (X3)	0,00 €
Potentiomètre 10kΩ	0,50 €
Plaque de soudure	5,00 €
Fils couleurs pour souder	Récupéré
Etain	1,00 €
Impression 3D	Info@Lèze
TOTAL:	**16,77 €**

La Production

La Soudure de la Plaque

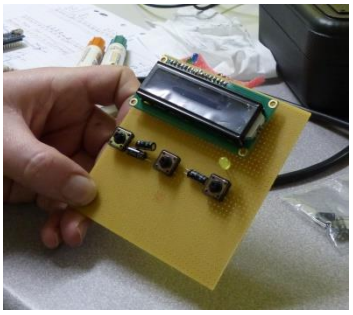
Les Schémas

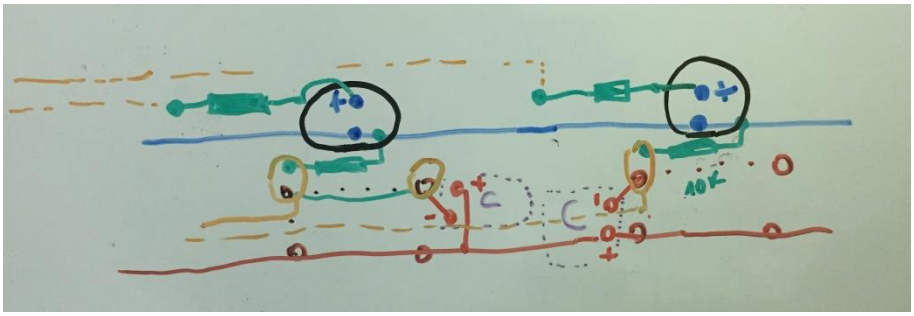
Avant de me lancer dans la soudure à l'étain, j'ai dû faire des schémas plus ou moins précis avec les emplacements des composants, et les câblages correspondants. Il fallait donc que j'ai une idée de la disposition des composants dans le boitier final avant de les souder.



Boîtier du Morscipio

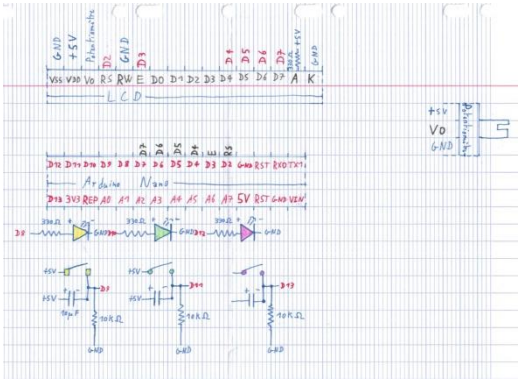
Le but étant, évidement, de limiter au maximum le nombre de soudures. Un des membres du club Info@Lèze m'a aidé à essayer de faire un montage avec un minimum de soudures : il semblait que faire une ligne de masse commune aux LEDs et boutons soit la meilleure solution.

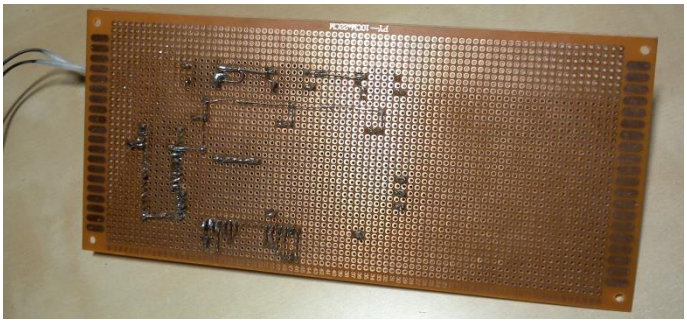




La Soudure

La soudure de la plaque se fait en deux étapes, d'abord, on soude les composants en respectant les schémas de montages, puis, on procède à la vérification des connections avant la mise sous tension. Cette deuxième étape est cruciale, car une erreur de soudure peut faire griller des composants, elle se fait à l'aide d'un multimètre muni d'une fonction permettant de vérifier si deux points sont connectés (le multimètre émet alors un signal sonore).

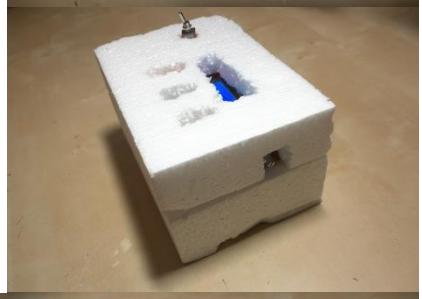




Le Boîtier

Choix du Matériau

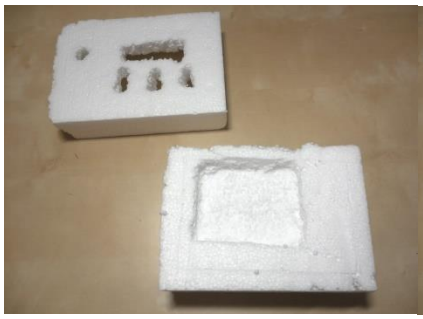
Pour améliorer la portabilité, ainsi que l'esthétique, il était évident qu'il fallait faire un boitier pour la plaque et les piles. D'abord, j'ai pensé à un boitier en bois, mais étant donné la complexité de la face avant, où il faudrait faire un trou pour chaque bouton/LED, un autre sur le côté pour le potentiomètre (réglage du contraste de l'écran LCD), un pour l'écran LCD, et un pour la sortie de la carte Arduino NANO, j'ai décidé d'en imprimer un en 3D. Comme le club Info@Lèze a une imprimante, ce serait facile.

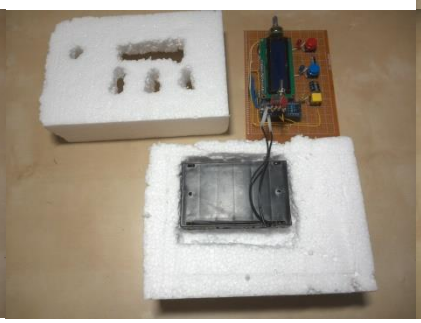


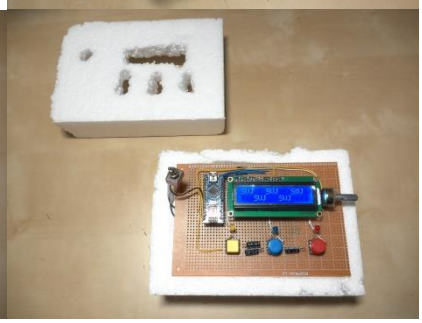
Essai en Polystyrène

Pour être sûr de ne pas oublier une ouverture sur le boîtier (par exemple, celle pour le potentiomètre…), j'ai choisi de faire un premier essai de boîtier en polystyrène. Ainsi, j'avais une première idée du rendu final. C'est en le manipulant que je me suis rendu compte qu'une ouverture pour pouvoir reprogrammer la carte serait utile. Ainsi, je pourrai le réutiliser pour un autre code que le morse, ou pour ajouter des chiffres et la ponctuation.

Je n'ai pas pu rendre compte de l'épaisseur du boîtier, car le polystyrène (récupéré) dont je disposais n'était pas de bonne qualité.



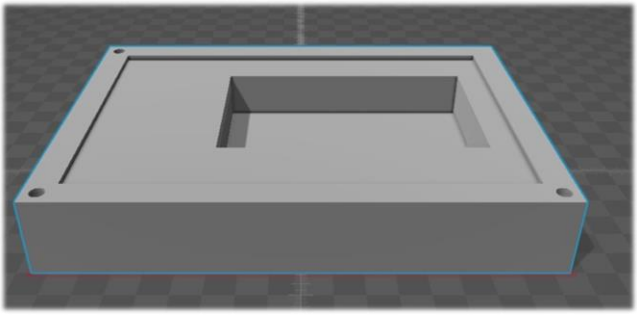


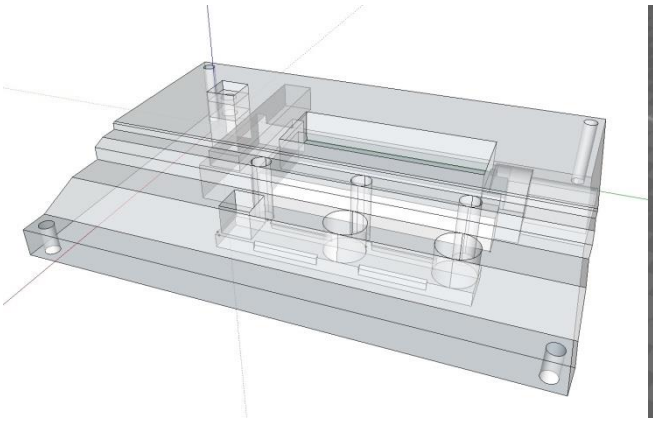


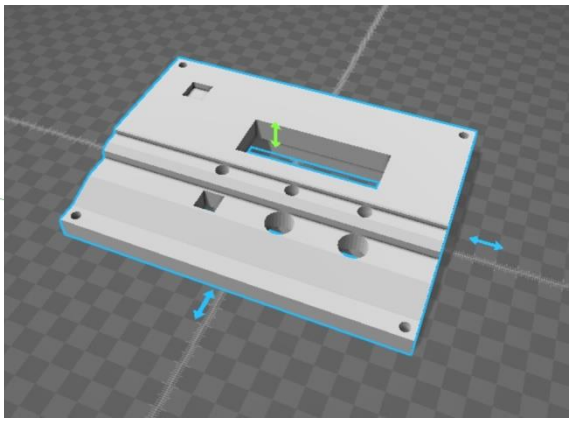
Conception / Dessin 3D

Pour pouvoir imprimer le boîtier en 3D, il fallait d'abord que je le dessine en 3D. Pour cela, j'ai utilisé le logiciel Sketch Up, qui est en licence libre (proposé par Google).

Il faut ensuite exporter le fichier en .stl pour pouvoir l'imprimer. Le format STL est l'équivalent du format PDF, mais pour les fichiers 3D.

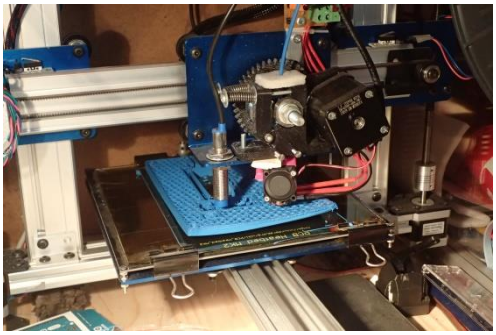


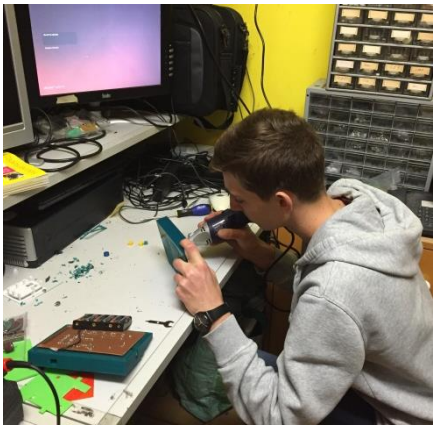


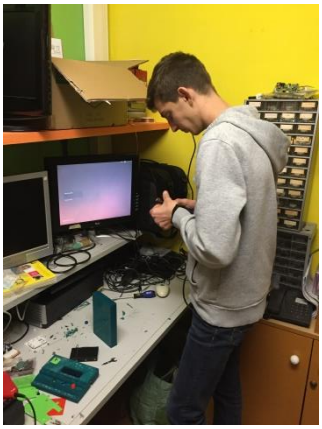


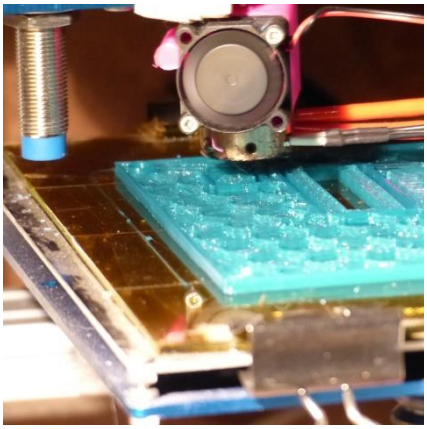
Impression en 3D

L'imprimante 3D a mis 5h à imprimer la partie du bas du boitier, et 7h pour celle du haut. Il a fallu régler des problèmes de warping (les côtés qui se soulèvent) en changeant de matériaux : l'ABS a été remplacé par du PLA.









Le Projet Fini !

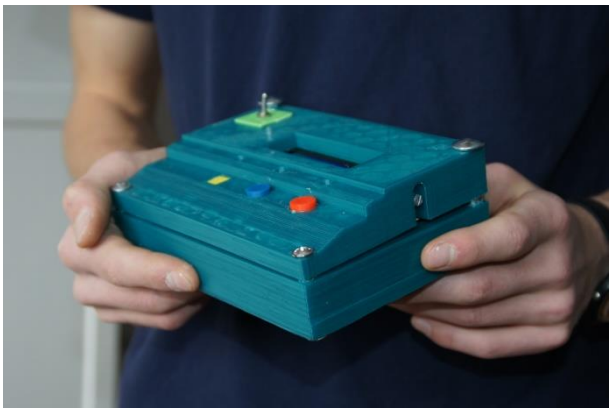
Après avoir tout emboité, rongé ce qui dépassait et vissé aux 4 coins (vis récupérées ☺), J'ai eu un bon résultat.

Le Nom : Morscipio

Après tout cela, il ne me restait plus qu'à trouver un nom pour ce boîtier décodeur de morse. J'ai donc cherché du côté de l'origine du mot « morse » mais il est dû au nom de son inventeur, Samuel Morse (comme Eugène Poubelle est à l'origine du mot « poubelle »). J'ai donc cherché l'origine du mot recevoir : cela vient du latin *recipio*, composé de *re-* et *cipio* qui veut dire prendre. C'est ainsi que j'ai trouvé le nom de Morscipio, contraction de *Morse* et de *cipio*, qui si on connait la racine est logique. D'autres noms m'étaient venu à l'idée (« Morsimple » ou « Boîte'au'Morse ») Morscipio est le plus accrocheur à mon goût.

Paul Dubois

Quelques photos en plus…





GAP