



## ESIPE-MLV

### ÉLECTRONIQUE ET INFORMATIQUE : SYSTÈMES COMMUNICANTS

#### COMPTE RENDU

EISC-1

---

## Projet Mini MasterMind

---

*Auteurs :*  
HÉRAT Clément  
MAGNI Paulu-Micheli

*Enseignants :*  
LOHIER Stéphane  
REVUZ Dominique

#### TP Programmation C

# Table des matières

<b>Introduction</b>	<b>1</b>
<b>1 Montage de l'écran LCD</b>	<b>2</b>
1.1 Présentation écran . . . . .	2
1.2 Montage . . . . .	4
<b>2 Ajout de deux boutons poussoirs</b>	<b>5</b>
2.1 Montage des boutons . . . . .	5
2.2 Montage Pull-Down . . . . .	6
2.3 Programme . . . . .	7
<b>3 Affichage et choix des caractères</b>	<b>10</b>
3.1 Choix de l'algorithme . . . . .	10
3.2 Programme . . . . .	10
<b>4 Calcul du résultat</b>	<b>14</b>
4.1 Algorithme . . . . .	14
4.2 Programme . . . . .	15
<b>5 Affichage du résultat</b>	<b>17</b>
5.1 L'affichage retenu . . . . .	17
5.2 Le programme . . . . .	18
<b>6 Synthèse</b>	<b>21</b>
6.1 Programme . . . . .	22
6.2 Quelques images de la version finale : . . . . .	33

# Table des figures

1	Mastermind . . . . .	1
1.1	Écran LCD . . . . .	2
1.2	Pin écran LCD . . . . .	2
1.3	Montage écran LCD . . . . .	4
2.1	Montage Boutons . . . . .	5
2.2	Montage d'une résistance en Pull-Down . . . . .	6
2.3	Test sans appuyer sur les boutons . . . . .	9
2.4	Test en appuyant sur un des deux boutons . . . . .	9
3.1	Début de la saisie de la 1 <sup>re</sup> réponse . . . . .	13
3.2	Fin de la saisie de la 5 <sup>e</sup> réponse . . . . .	13
5.1	Exemples de résultats obtenus . . . . .	17
6.1	Montage final avec batterie pour pouvoir jouer partout . . . . .	33
6.2	Montage final du Projet Mastermind . . . . .	34
6.3	Fin de partie . . . . .	35

# Introduction

Dans le cadre de nos cours de programmation en langage C, et de leurs application en Travaux Dirigés sur Arduino<sup>1</sup>, nous avons du réaliser un MasterMind.



FIGURE 1 – Mastermind

Pour ceux qui ne connaissent pas, le MasterMind est un jeu de réflexion et de déduction qui se joue à deux. Nous le trouvons généralement sous la forme de la figure1.

En voici donc le principe d'après Wikipedia : Un joueur commence par placer son choix de pions sans qu'ils soient vus de l'autre joueur à l'arrière d'un cache qui les masquera à la vue de celui-ci jusqu'à la fin de la manche. Le joueur qui n'a pas sélectionné les pions doit trouver quels sont les quatre pions, c'est-à-dire leurs couleurs et positions.

Pour cela, à chaque tour, le joueur doit se servir de pions pour remplir une rangée selon l'idée qu'il se fait des pions dissimulés. Une fois les pions placés, l'autre joueur indique :

1. le nombre de pions de la bonne couleur bien placés en utilisant le même nombre de pions rouges.
2. le nombre de pions de la bonne couleur, mais mal placés, avec les pions blancs.

Il arrive donc surtout en début de partie qu'il ne fasse rien concrètement et qu'il n'ait à dire qu'aucun pion ne correspond, en couleur ou en couleur et position. La tactique du joueur actif consiste à sélectionner en fonction des coups précédents, couleurs et positions, de manière à obtenir le maximum d'informations de la réponse du partenaire puisque le nombre de propositions est limité par le nombre de rangées de trous du jeu.

Dans la plupart des cas, il s'efforce de se rapprocher le plus possible de la solution, compte tenu des réponses précédentes, mais il peut aussi former une combinaison dans le seul but de vérifier une partie des conclusions des coups précédents et de faire en conséquence la proposition la plus propice à la déduction d'une nouvelle information.

Le joueur gagne cette manche s'il donne la bonne combinaison de pions sur la dernière rangée ou avant. Dans tous les cas, c'est à son tour de choisir les pions à découvrir. Mais il est interdit de mettre une couleur en double, en triple ou en quadruple aussi bien dans les pions secrets que dans les pions « publics ».

Les objectifs de ce projet sont :

- de développer des algorithmes.
- d'appliquer les connaissances théoriques acquises en cours de programmation de langage C.
- de travailler sur les entrées/sorties de la carte Arduino.
- de réaliser un montage électronique fonctionnel.

---

1. Arduino is an open-source electronics platform based on easy-to-use hardware and software. It's intended for anyone making interactive projects. <https://www.arduino.cc/>

# Chapitre 1

## Montage de l'écran LCD

### 1.1 Présentation écran

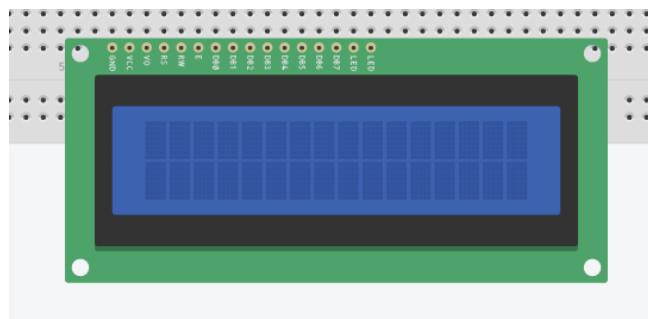


FIGURE 1.1 – Écran LCD

Au cours de ce projet nous allons utiliser un écran LCD 16x2 équipé du driver Hitachi HD44780 (figure 1.1). Ce type d'écran est compatible avec la librairie « LiquidCrystal » disponible nativement au sein de l'IDE<sup>1</sup> Arduino

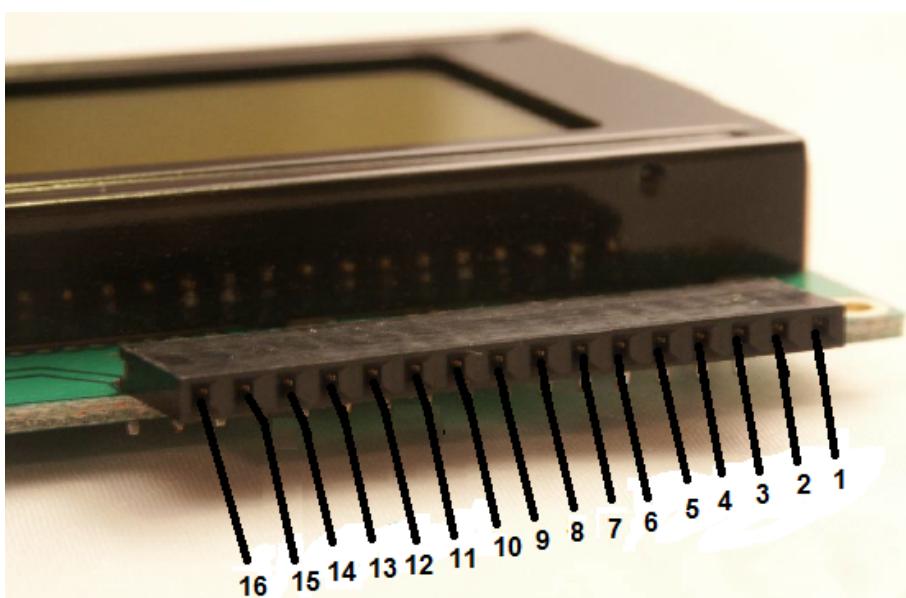


FIGURE 1.2 – Pin écran LCD

---

1. IDE : Entegrated Eevelopment Environment ; ou EDI en français : Environnement de Développement Intégré.  
source Wikipedia

Description des pins :

- 1 Vss - Masse 0V de l'écran LCD.
- 2 Vcc - Pin d'alimentation de l'écran. Le voltage accepté est entre 3.3V et 5V.
- 3 Pin d'ajustement du contraste. On y branche un potentiomètre pour faire varier le contraste.
- 4 Pin sélection du registre. Si égale à 0 : mode commande, si égale à 1 : mode données.
- 5 Pin de lecture/écriture. Si égale à 0 : mode écriture, si égale à 1 : mode lecture.
- 6 Pin de validation.
- 7 Bus de données, bit 0
- 8 Bus de données, bit 1
- 9 Bus de données, bit 2
- 10 Bus de données, bit 3
- 11 Bus de données, bit 4
- 12 Bus de données, bit 5
- 13 Bus de données, bit 6
- 14 Bus de données, bit 7
- 15 Alimentation du rétroéclairage de l'écran. La tension d'entrée maximale est 5V et correspond à l'éclairage maximum. On peut y connecter un potentiomètre pour faire varier l'intensité du rétroéclairage.
- 16 Pin de terre du rétroéclairage.

## 1.2 Montage

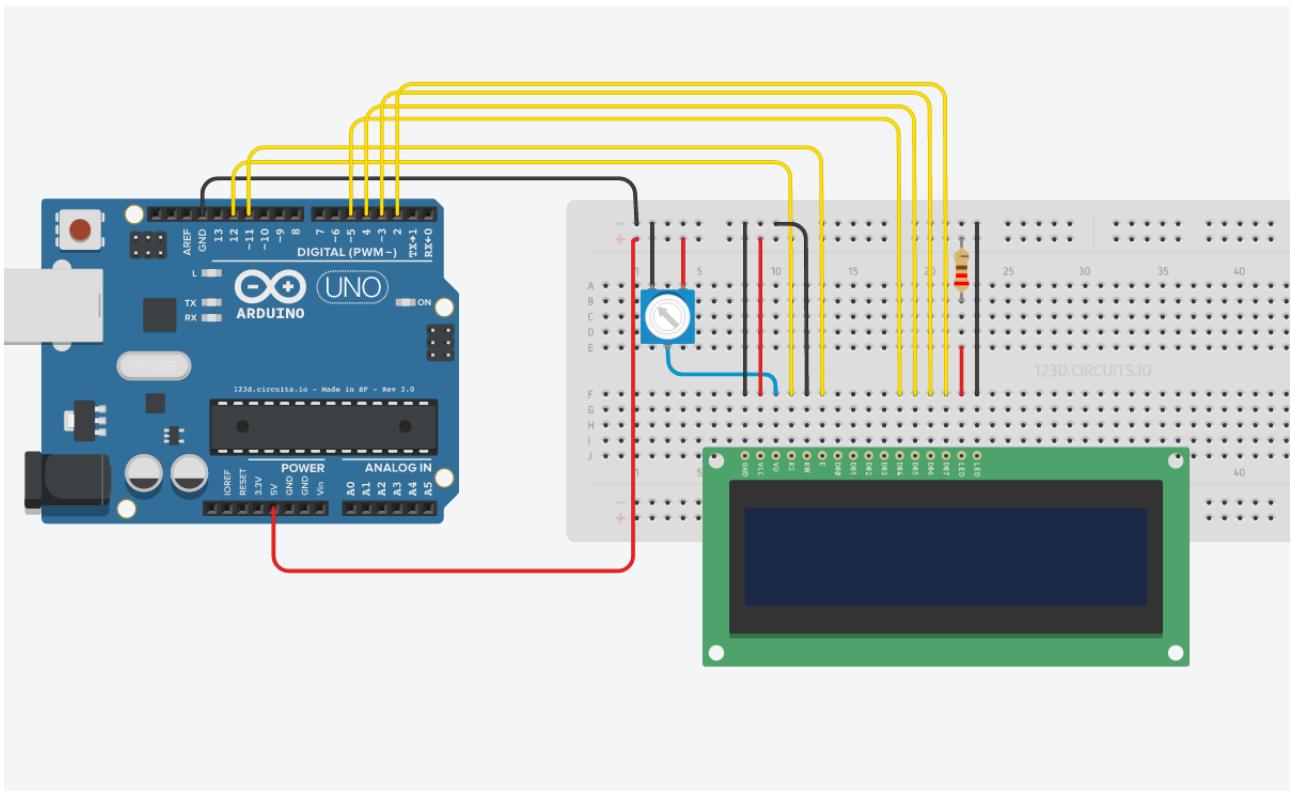


FIGURE 1.3 – Montage écran LCD

Pour le montage de l'écran LCD et de l'Arduino (figure 1.3) nous avons utilisés :

- 1 écran LCD,
- 1 potentiomètre  $10k\Omega$ ,
- 1 Arduino Uno,
- 1 résistance  $220\Omega$ ,
- 5 fil noir,
- 4 fil rouge,
- 1 fil bleu,
- 6 fil jaune.

# Chapitre 2

## Ajout de deux boutons poussoirs

### 2.1 Montage des boutons

Comme vous pouvez le voir sur la figure 2.1 nous avons montés les boutons en montage Pull-Down ainsi que évoqué sur la figure 2.2 sur les entrées 8 et 9 de l'Arduino.

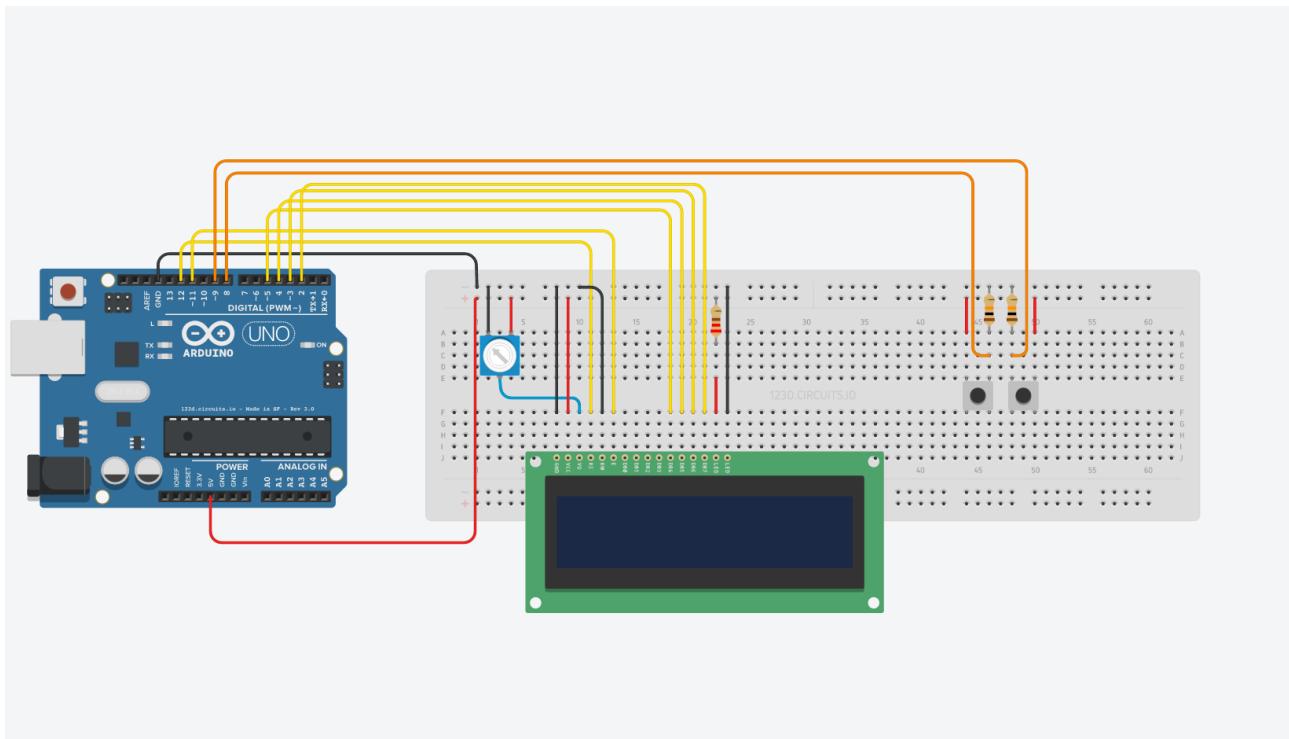


FIGURE 2.1 – Montage Boutons

## 2.2 Montage Pull-Down

Nous pouvons remarquer sur le montage de la figure 2.1 deux résistance placées à la sortie des boutons poussoir. Il s'agit de résistance dites "Pull-Down". Ces résistances ont deux rôles très importants, elles servent à assurer le bon fonctionnement du bouton, mais également la sécurité du circuit.

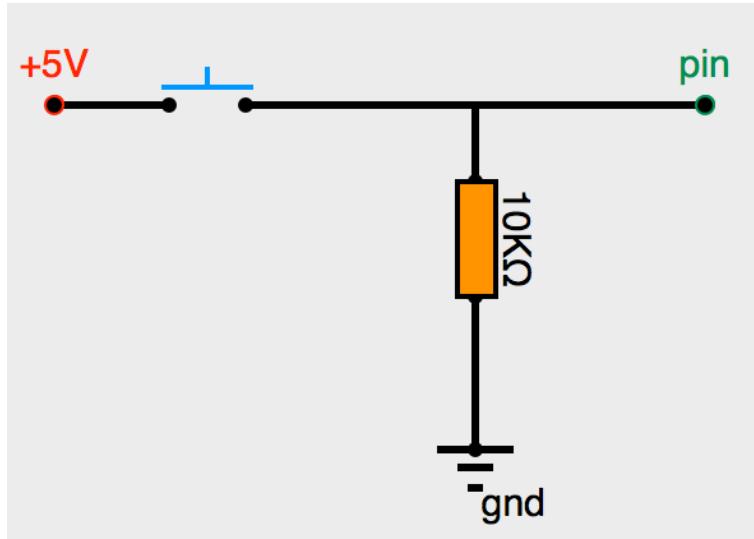


FIGURE 2.2 – Montage d'une résistance en Pull-Down

L'électricité choisit toujours le chemin le moins résistant. Mais si elle n'a pas le choix, elle passe tout de même là où ça résiste.

Nous allons donc ajouter une résistance à notre circuit. Une assez forte pour que le courant ne passe que s'il y est obligé (ici de  $10k\Omega$ ). Si le poussoir est baissé, le courant va du +5V à la pin de l'Arduino. Il ne prendra pas le chemin de la masse car la résistance lui demande un effort (Et dans le cas contraire la résistance protégera le montage d'un court-circuit). La pin de l'Arduino recevra du +5V et indiquera HIGH (ou 1).

Si le poussoir est levé, donc le circuit ouvert, le très faible courant résiduel qui sortira du pin de l'Arduino sera absorbé par la masse, le pin sera donc bien en LOW (ou 0)

## 2.3 Programme

Programme utilisé pour tester le bon fonctionnement de l'acquisition des boutons :

```
/**  
 * *****TEST*****  
 * *****  
 * Programme de test pour la saisie avec les 2 boutons  
 */  
  
#include <LiquidCrystal.h>  
  
/**  
 * Initialisation de la librairie avec les sorties de l'Arduino utilisées pour l'écran  
 * Et déclaration des entrées de l'Arduino pour les boutons  
 */  
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  
const int button_1 = 8;  
const int button_2 = 9;  
  
/**  
 * Initialisation de l'Arduino :  
 * indiquer à la librairie les "dimensions" de l'écran  
 * indiquer le mode de fonctionnement des pins pour les boutons  
 */  
void setup() {  
    lcd.begin(16, 2);  
    pinMode(button_1, INPUT);  
    pinMode(button_2, INPUT);  
}  
  
/**  
 * Fonction d'acquisition de l'état des boutons  
 * On attend que l'utilisateur appuie sur un bouton et lorsqu'il le fait on retourne le  
 * chiffre correspondant  
 *  
 * @switchState [int] est une variable qui nous sert à stocker le numéro du bouton  
 * allumé.  
 * @return un entier qui correspond au bouton qui est allumé  
 */  
int GetButtons(){  
    int switchState = 0;  
    while((digitalRead(button_1) != HIGH) && (digitalRead(button_2) != HIGH) ){}  
  
    if (digitalRead(button_1) == HIGH)  
        switchState = 1;  
  
    if (digitalRead(button_2) == HIGH)  
        switchState = 2;  
  
    return switchState;  
}  
  
/**  
 * Boucle principale du programme  
 *  
 * on va en permanence afficher "James"  
 * et lorsqu'on appuiera sur un des boutons on affichera "Bond" à la suite de "James"  
*/
```

```
 */
void loop() {
    lcd.clear();

    lcd.print("James");

    delay(200);

    if (GetButtons()) {
        lcd.print("Bond");
        delay(200);
    }

    lcd.clear();
}
```

Voici les affichages obtenus :

- Lorsqu'on appuie sur aucun boutons :

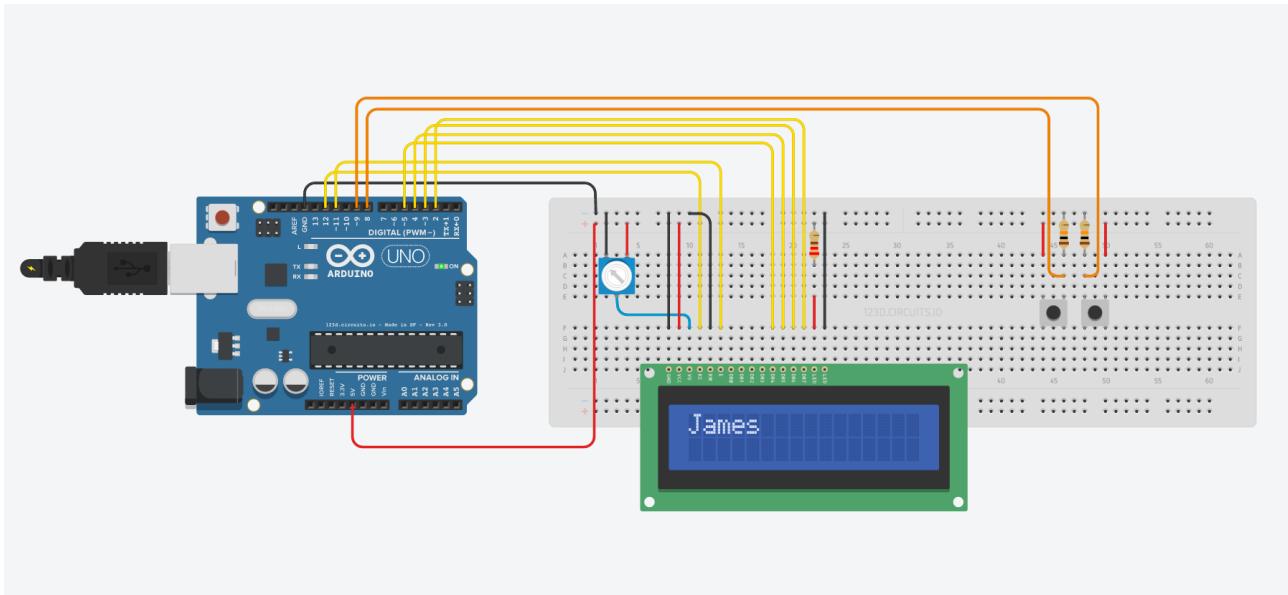


FIGURE 2.3 – Test sans appuyer sur les boutons

- Lorsqu'on appuie sur un des 2 boutons :

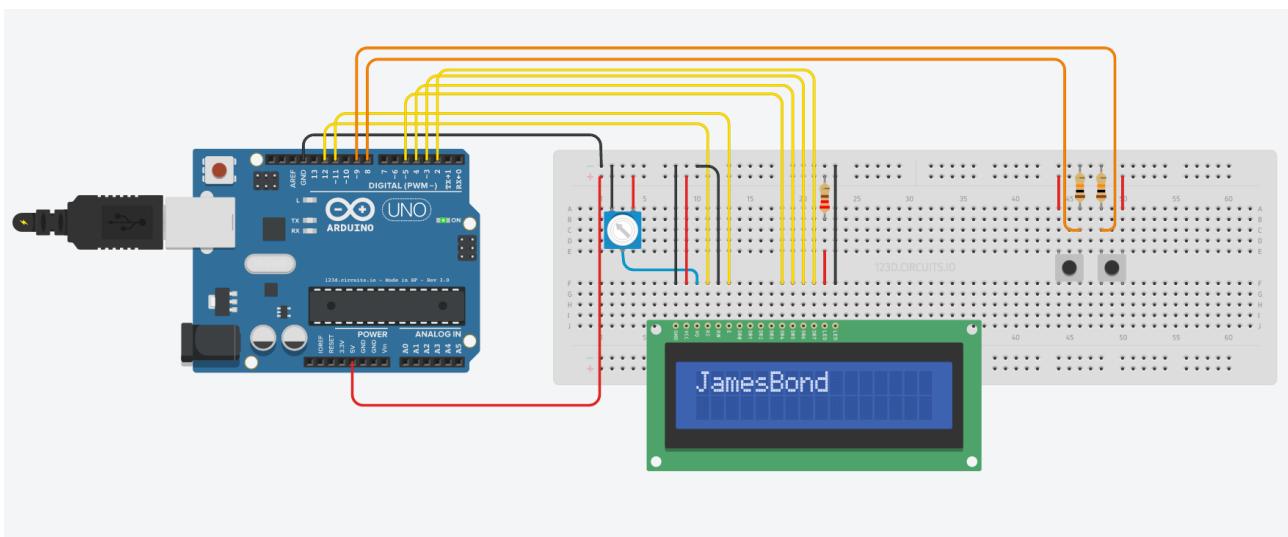


FIGURE 2.4 – Test en appuyant sur un des deux boutons

# Chapitre 3

## Affichage et choix des caractères

Il faut maintenant réaliser une fonction pour l'affichage et le choix des caractères afin que le joueur puisse saisir une réponse.

Il faut également que cette fonction stocke, de façon pertinente, la réponse du joueur afin qu'une autre fonction puisse en calculer le résultat. C'est à dire comparer sa réponse à la solution et dire combien de lettres sont bonnes mais mal positionnées et combien de lettres sont bonnes et bien placées. Et qu'entre deux réponses nous ayons la place d'afficher le résultat de la réponse précédente.

### 3.1 Choix de l'algorithme

Pour réaliser cette fonction nous avons identifiés plusieurs solution lors de l'étude préalable du programme. Nous avons donc choisi de réaliser une fonction qui enregistrera seulement la réponse actuellement en cours de saisie et qui utilisera la rémanence de l'écran pour conserver les réponses et résultats précédents.

En effet précédemment nous avions remarqué que tant que l'écran n'était pas "nettoyé" l'affichage se conservait dans son état actuel. Cette solution simplifiait beaucoup le reste du programme et permettait de limiter les accès mémoire lors de la saisie des réponses et de l'affichage.

### 3.2 Programme

Programme utilisé pour tester la fonction de saisie :

```
/**  
 * *****TEST*****  
 * *****  
 * Programme de test pour la fonction de saisie des réponses  
 */  
  
#include <LiquidCrystal.h>  
  
/**  
 * Initialisation de la librairie avec les sorties de l'Arduino utilisées pour l'écran  
 * Et déclaration des entrées de l'Arduino pour les boutons  
 */  
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  
const int button_1 = 8;  
const int button_2 = 9;  
  
/**  
 * Initialisation de l'Arduino :  
 * indiquer à la librairie les "dimensions" de l'écran  
 * indiquer le mode de fonctionnement des pins pour les boutons  
 */
```

```

void setup() {
    pinMode(button_1, INPUT);
    pinMode(button_2, INPUT);

    lcd.begin(16, 2);
}

/***
 * Fonction d'acquisition de l'état des boutons
 * On attend que l'utilisateur appuie sur un bouton et lorsqu'il le fait on retourne le
 * chiffre correspondant
 *
 * @switchState [int] est une variable qui nous sert a stocker le numero du bouton
 * allumé.
 * @return un entier qui correspond au bouton qui est allumé
 */
int GetButtons(){
    while((digitalRead(button_1) != HIGH) && (digitalRead(button_2) != HIGH) ){}

    if (digitalRead(button_1) == HIGH)
        return 1;

    if (digitalRead(button_2) == HIGH)
        return 2;

    else
        return 0;
}

/***
 * Fonction d'acquisition de la réponse du joueur
 *
 * Nous faisons appel a la fonction GetButtons(...) pour obtenir les infor , afin de
 * récupérer l'information des boutons.
 *
 * Pour chacun des 4 éléments de sa réponse le joueur va faire défiler les
 * lettres possibles avec le bouton 1.
 *
 * A chaque fois qu'il appuie sur le bouton 1, on va afficher la lettre
 * actuellement choisie.
 *
 * Il va ensuite valider la lettre choisie avec le bouton 2. Et le curseur
 * va se déplacer d'une case sur l'écran.
 *
 * Chaque lettre choisie sera stockée à la place correspondante dans le
 * tableau "playerAnswer"
 *
 * Nous utilison la table ASCII pour les lettres. Ce qui permet une manipulation
 * plus simple. En effet pour passer a la lettre suivante il suffit d'incrémenter
 * sa valeur de 1 et de faire un modulo afin de ne pas sortir de l'ensemble de
 * lettre possible (de A à F ici en l'occurence).
 *
 * @param playerAnswer [char *] Tableau de 4 char contenant la réponse du joueur
 * @param dx           [int *] Pointeur qui contient la position en x du curseur
 *                      sur l'écran
 * @param dy           [int *] Pointeur qui contient la position en y du curseur
 *                      sur l'écran
 */
void InputLetters (char *playerAnswer, int *dx, int *dy){
    int input = 0;
    int letter = 0;

    for(;letter <= 3;){
        lcd.setCursor(*dx,*dy);

```

```

lcd.write(input+65);

if (GetButtons() == 1){
    input = (input + 1) % 6;
}
if (GetButtons() == 2){
    playerAnswer[letter] = input + 65;
    input = 0;
    letter++;
    *dx += 1;
}

delay(200);
}

/***
 * Boucle principale du programme
 *
 * Nous allons effacer l'écran en 1er lieu afin d'éviter tout reste d'affichage
 *
 * Ensuite nous plaçons le curseur à l'origine de l'écran (home = position(0,0))
 *
 * Nous laissons 6 tentatives de réponse au joueur.
 * A chaque tentative nous faisons appel à la fonction InputLetters(...) pour
 * l'acquisition de la réponse.
 *
 * Lorsqu'une tentative de réponse a été faite nous faisons 2 choses :
 * - on regarde si le curseur est proche du bord droit de l'écran
 *   (position en x supérieur à 13) et si oui on passe à la ligne en
 *   incrémentant la position en y de 1 en vérifiant qu'on ne sort pas non
 *   plus de l'écran.
 * - on incrémente la position du curseur en x de 1 afin de sauter une case
 *   d'affichage qui nous servira plus tard pour afficher les résultats de la
 *   tentative.
 *
 */
void loop() {
    lcd.clear();
    lcd.home();
    char playerAnswer[4];
    int dx = 0;
    int dy = 0;
    int nbTry;
    for (nbTry = 0; nbTry < 6; nbTry++) {
        InputLetters(playerAnswer, &dx, &dy);
        if (dx >= 13)
            dy = (dy + 1) % 2;

        dx = (dx + 1) % 15;
    }
    delay(1000);
}

```

Affichage obtenus lors du test de l'affichage et de la saisie des caractères :

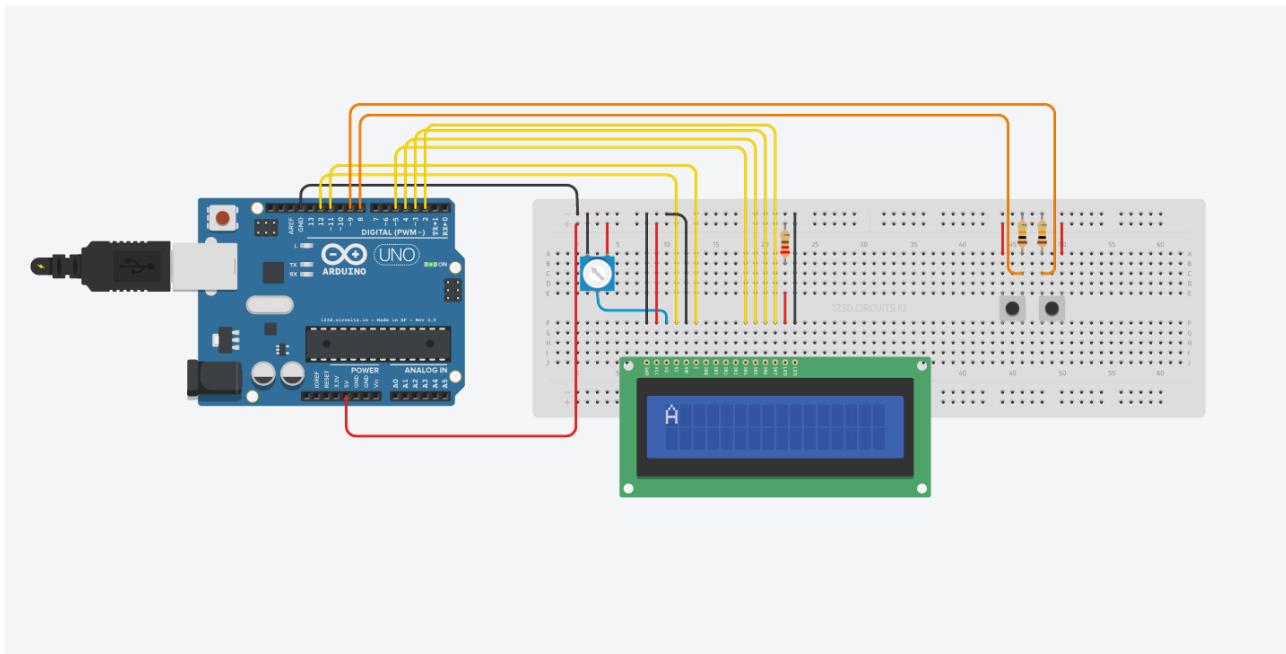


FIGURE 3.1 – Début de la saisie de la 1<sup>re</sup> réponse

Comme nous pouvons le voir sur la figure 3.2 l'affichage est conforme à nos attentes, nous avons bien un espace vide entre deux réponses ainsi qu'un saut à la ligne arrivé au bord de l'écran.

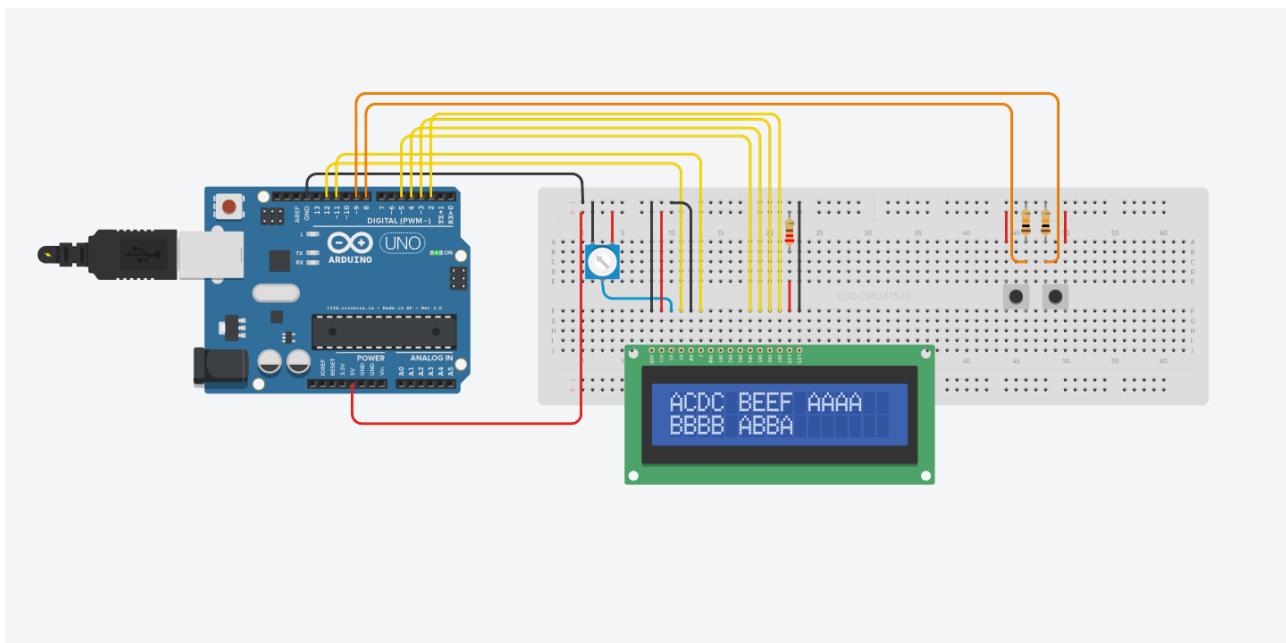


FIGURE 3.2 – Fin de la saisie de la 5<sup>e</sup> réponse

# Chapitre 4

## Calcul du résultat

Il faut maintenant réaliser une fonction de calcul du résultat. Cette fonction devra à partir du code secret et de la réponse entrée par le joueur nous indiquer le nombre de lettres bien placées et le nombre de lettres mal placées.

### 4.1 Algorithme

Pour cet algorithme il a fallut prendre en compte un point important : la possibilité d'avoir plusieurs fois la même lettre dans le code ou la réponse. Un algorithme ne prenant pas cette information en compte pourrait entraîner des erreurs.

Prenons un exemple :

- Code secret : "ABBA"
- Réponse : "ABCD"

Dans ce cas un algorithme ne prenant pas en compte les double lettres pourraient considérer le A de la réponse comme étant à la fois à la bonne place mais aussi comme étant mal placé car "ABBA" contient deux 'A'. Pour éviter cela nous avons du mettre en place un moyen de mémoriser les éléments déjà identifiés comme bien ou mal placés afin d'éviter le problème de double résultats.

Nous avons donc décidé de directement modifier les tableaux contenant le code secret et la réponse.

Lorsque nous identifions deux lettres comme étant identiques :

- La lettre du tableau code secret est passées en minuscules afin d'empêcher une autre égalité lors de la comparaison sans perdre de donnée car il nous suffira simplement de repasser les lettres en majuscules pour revenir à l'état initial
- La lettre du tableau réponse est mise à 1 ou 2 selon qu'elle soit bien placée ou non. La perte de donnée n'étant pas importante car nous ne réutilisons pas ces données par la suite.

Une fois cela pris en compte l'algorithme est assez simple et se déroule en trois étapes :

1. identification des lettres bien placées
2. identification des lettres mal placées
3. mise en majuscule des lettres ayant été mises en minuscule

## 4.2 Programme

Programme utilisé pour calculer les résultats :

```
#include stdio.h
#include stdlib.h

/*Fonction de calcul du résultat.*/

/*
Le calcul du résultat s'effectue en deux temps
1. Détermination des lettres bien placées
Pour cela nous comparons les lettres situées aux même positions (user[i] avec
secret[i]) et en cas d'égalité
- on incrémente la case du tableau result correspondant au lettres bien placée
- afin d'éviter un doublon dans les recherche suivantes on met la lettre du
tableau secret en minuscule cela nous permettant également de ne pas perde
le mot secret
- toujours dans le but d'éviter une double occurrence pour les prochaine
recherche on met la lettre du tableau user a la valeur 1 (nous pouvons é
craser cette donnée car nous ne la réutiliserons plus)
2. Détermination des lettres mal placées
Pour cela nous procérons de la même manière que précédemment mais cette fois ci en
comparant chaque lettre du tableau user a chaque lettre du tableau secret
nous remettons ensuite les lettre du tableau secret en majuscule
*/
int GetAnswer(char secret, char user){

    int i, j; /* Variable pour le parcours des tableaux */

    int result = (int)malloc(sizeof(int)*2); /*Tableau de Résultat la première(0) case
        correspond au éléments bien placés et la deuxième(1) aux éléments mal placés*/
    result[0] = 0; result[1] = 0;

    for (i = 0; i < 4; ++i)/*Détermination des lettres bien placées*/
    {
        if (user[i] == secret[i])
        {
            result[0] += 1; /*Incrémantation du nombre de lettres bien placées*/
            secret[i] += 32; /*Mise en minuscule de la lettre*/
            user[i] = '1'; /*Mise a 1 de la lettre*/
        }
    }

    if (result[0] != 4)/*Si toutes les lettres sont bien placées on ne vérifie pas si
        il y en a des mal placées*/
    {
        for (i = 0; i < 4; i++)/*Détermination des lettres bien placées*/
        {
            for (j = 0; j < 4; j++)
            {
                if (user[i] == secret[j])
                {
                    result[1] += 1; /*Incrémantation du nombre de lettres bien placées*/
                    secret[j] += 32; /*Mise en minuscule de la lettre*/
                    user[i] = '2'; /*Mise a 2 de la lettre*/
                }
            }
        }
    }

    for (int i = 0; i < 4; ++i)/*mise en majuscule des lettre ayant été mise en
        minuscule*/
}
```

```

{
    if ( secret [ i ] == 96)
    {
        secret [ i ] = secret [ i ] == 32;
    }
}

return result;
}

int main( int argc , char const argv [] )
{
    char secret [ 4 ] = ABBA;
    char user [ 4 ] = ACDC;
    int i;

    int tab = GetAnswer( secret , user );

    for ( i = 0; i < 4; ++i )/*Test du résultat*/
    {
        printf( "%c" , secret [ i ] );
    }
    printf( "\n" );

    for ( i = 0; i < 4; ++i )/*Test du résultat*/
    {
        printf( "%c" , user [ i ] );
    }
    printf( "\n" );

    for ( int i = 0; i < 2; ++i )/*Test du résultat*/
    {
        printf( "%d" , tab [ i ] );
    }

    return 0;
}

```

# Chapitre 5

# Affichage du résultat

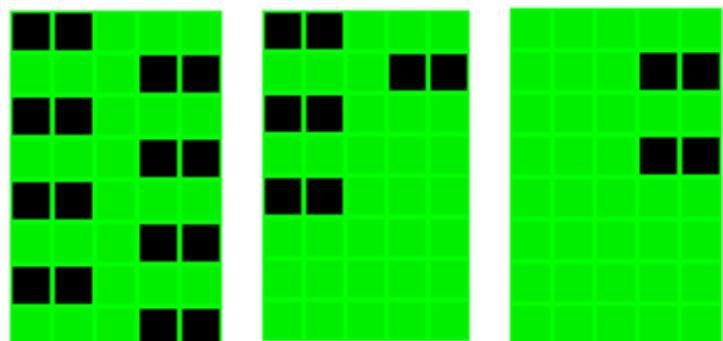
Après avoir réalisé la fonction de calcul du résultat il convient d'en réaliser sa fonction d'affichage. Nous avons trouvé que la méthode d'affichage en binaire proposée dans le cadre du projet n'était pas forcément des plus intuitive à comprendre.

En conséquence nous en avons cherché une nouvelle plus facile à comprendre pour un néophyte.

## 5.1 L'affichage retenu

Comme nous sommes sur un afficheur où chaque caractères est constitué de 5x8 pixels, nous avons choisi l'affichage que vous pouvez voir sur la figure 5.1a avec sur la gauche autant de lignes que d'éléments bien placés et à droite autant de lignes que de bons éléments mais mal placés

Pour davantage de facilités à lire et à implémenter également, nous avons disposés les lignes alternativement.



(a) Affichage de toutes les possibilités

(b) Dans le cas où on en a 3 bien placés et 1 mal placé

(c) Dans le cas où on en a 0 de bien placés et 2 de mal placé



(d) En situation de jeu dans le cas où la réponse secrète est DBAE

FIGURE 5.1 – Exemples de résultats obtenus

## 5.2 Le programme

Pour parvenir au résultat ci-dessus nous avons utilisé la fonction « createChar()<sup>1</sup> » de la librairie LiquidCrystal. Pour faire un bref résumé, c'est une fonction qui permet de créer un caractère pixel par pixel. A cette fin il y a 8 caractères de la librairie qui sont disponibles (de 0 à 8). Ceci est possible car dans la librairie LiquidCrystal les caractères sont sous la forme suivante :

```
byte smiley[8] = {  
    B00000,  
    B10001,  
    B00000,  
    B00000,  
    B10001,  
    B01110,  
    B00000,  
};
```

C'est un tableau de type byte où chaque « B\*\*\*\*\* »(byte) définit une ligne, chaque « \* » peut valoir 0 ou 1. On peut donc éteindre ou allumer chaque pixels de chaque lignes indépendamment.

Programme d'affichage des résultats :

```
/**  
 * *****TEST*****  
 * Programme de test pour la fonction de saisie des réponses  
 */  
  
#include <LiquidCrystal.h>  
  
/**  
 * Initialisation de la librairie avec les sorties de l'Arduino utilisées pour l'écran  
 * Et déclaration des entrées de l'Arduino pour les boutons  
 */  
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  
const int button_1 = 8;  
const int button_2 = 9;  
  
/**  
 * Initialisation de l'Arduino :  
 * indiquer à la librairie les "dimensions" de l'écran  
 * indiquer le mode de fonctionnement des pins pour les boutons  
 */  
void setup() {  
    pinMode(button_1, INPUT);  
    pinMode(button_2, INPUT);  
    lcd.begin(16, 2);  
}  
  
/**  
 * Fonction de l'affichage des résultats  
 *  
 * Le but de cette fonction est de réaliser le caractère spécial correspondant  
 * aux résultats de la réponse précédente du joueur.  
 *  
 * Nous commençons par créer un tableau de bytes éteint pour le cas où il n'y a aucune  
 * bonnes réponses  
 *  
 * Ensuite nous modifions ce tableau de byte éteint en y remplaçant un élément par  
 * B11000 (ligne gauche, bien placés) ou B00011 (ligne droite, mal placés) en fonction  
 * des résultats  
 */
```

```

* Puis nous créons le caractère spécial désiré avec ce tableau de byte
*
* enfin nous plaçons le curseur de l'écran lcd à la bonne position et on affiche
* le caractère créé en veillant à ce qu'il ne sorte pas de la zone d'affichage
*
*
* @param results [tableau de int] results[0] est le nombre d'éléments bien placés
* @param dx      [int *] Pointeur qui contient la position en x du curseur
*                 sur l'écran
* @param dy      [int *] Pointeur qui contient la position en y du curseur
*                 sur l'écran
* @param nbTry   [int] numéro de la tentative qui sert a definir un nouveau
*                     caractère spécial correspondant a la réponse que nous
*                     voulons afficher.
*/
void Printer(int *results, int *dx, int *dy, int nbTry) {
    /** Déclaration caractère vide */
    byte figure[8] = {B00000,B00000,B00000,B00000,B00000,B00000,B00000,B00000};

    /**
     * On va modifier ce tableau afin qu'il corresponde aux résultats s'il y a des
     * éléments bien ou mal placés
     */
    if((results[0] != 0) || (results[1]!=0)){
        /**
         * p est une variable qui va nous permettre de nous déplacer dans le tableau
         * de bytes créé précédemment
         */
        int p =0;

        /**
         * on va s'occuper de créer les lignes correspondante au nombre d'éléments
         * bien placés
         */
        for(int i = 0; i<results[0]&& p<=9; i++){
            figure[p]=B11000;
            p+=2;
        }
        /**
         * on définit p à 1 afin d'avoir le décalage entre les lignes de bien placés
         * et les lignes de mal placés
         */
        p=1;
    }

    /**
     * Et ici on va créer les lignes correspondant au nombre d'éléments mal placés
     */
    for(int i = 0; i<results[1] && p<=9; i++){
        figure[p]=B000011;
        p+=2;
    }
}

/**
 * On s'occupe de créer le caractère spécial à partir du tableau figure
 * modifié (ou pas) précédemment et de réaliser l'affichage au bon endroit sur
 * l'écran LCD
*/
lcd.createChar(nbTry+1, figure);
lcd.setCursor(*dx,*dy);
lcd.write(nbTry+1);

/**
 * on veille ici a ne pas sortir de la zone d'affichage de l'écran LCD
*/

```

```

if ((*dx+1)==15){
    *dy=(*dy+1);
}
*dx=(*dx+1)%15;

delay(600);

}

/***
 * Boucle principale du jeu
 *
 * Nous allons déclarer en dur le tableau results qui contient le nombre de
 * bien placés et le nombre de mal placés afin de tester la fonction
 * d'affichage des résultats
 */
void loop() {
    int result[2]={1;2};
    int dx = 0;
    int dy = 0;

    Printer(results, &dx, &dy, 1)
    delay(10000);
}

```

---

1. voir la page :[arduino.cc](http://arduino.cc)

# Chapitre 6

## Synthèse

Dans cette version 2.1 du Mastermind vous trouverez des leds mais aussi des niveaux, cinq en tout (extrait du commentaire final du programme) :

1. *Beginner* : niveau débutant, "finger in the nose"
  - Pas le droit à plusieurs fois la même lettre
  - Les lettres possible sont de A à E
2. *Normal* : règles officielle du Mastermind
  - Pas le droit à plusieurs fois la même lettre
  - Les lettres possible sont de A à F
3. *Teacher* : règles demandées dans le sujet du projet
  - Plusieurs même lettres possibles
  - Les lettres possible sont de A à F
4. *Hard* : niveau difficile, "Good Luck!"
  - Plusieurs même lettres possibles
  - Les lettres possible sont de A à G
5. *42* : niveau solution 404 "Fuyez Pauvres Fous!"
  - Plusieurs même lettres possibles
  - Les lettres possible sont tout l'alphabet !!

## 6.1 Programme

Code source du programme :

```
/****************************************************************************
 * *****MASTERMIND*****
 * ****
 * ##### Version 2.1 #####
 * ***
 * Authors:      C.HERAT && P-M.MAGNI
 * ***
 * last edit: 06/02/2017
 * ****
 */

#include <LiquidCrystal.h>


```

```

/*
void ResetAnswer( char *Answer){
    for( int i = 0; i<4; i++){
        Answer[ i ] = '\0';
    }
}

/***
* Pour faire les différents affichages avec un texte qui défile du programme,
* nous avons créé cette fonction qui prend un texte en entrée et l'affiche
* sur l'écran en le faisant se décaler vers la gauche
*
* @param text [chaine de caractères] c'est le texte à afficher.
*/
void ScrollText(char *text){
    int len = strlen(text);

    // set the cursor to (16,0):
    lcd.setCursor(16,0);
    // set the display to automatically scroll:
    lcd.autoscroll();
    for (int i = 0; i < len; i++) {
        lcd.print(text[i]);
        delay(200);
    }
    lcd.noAutoscroll();
    lcd.clear();
}

/***
* Permet de faire clignoter un texte sur l'écran lcd.
*
* @param text [chaine de caractère à faire clignoter
*/
void BlinkLcd(char *text){
    lcd.write(text);
    for( int i = 0; i<4 ; i++){
        // Turn off the display:
        lcd.noDisplay();
        delay(300);
        // Turn on the display:
        lcd.display();
        delay(200);
    }
    //lcd.clear();
}

/***
* Permet de faire clignoter les leds au lancement du jeu
*
* @param flash_num      [int] indique le nombre de clignotement
* @param flash_speed    [int] indique la vitesse de clignotement
*/
void BlinkLed(int flash_num , int flash_speed){
    for( int i = 0; i<flash_num; i++){
        digitalWrite(led_r , HIGH);
        digitalWrite(led_v , HIGH);
        delay(flash_speed);
        digitalWrite(led_r , LOW);
        digitalWrite(led_v , LOW);
        delay(flash_speed);
}

```

```

        }

    /**
     * Permet de faire clignoter de façon particulière les leds lors des défaites ou
     * des victoires
     *
     * @param led      [int] définit la led à faire clignoter
     * @param flash_speed [int] définit la vitesse de clignotement
     * @param flash_num   [int] définit le nombre de clignotement
     */
void LedAnswer(int led, int flash_speed, int flash_num){
    for(int i=0; i<flash_num; i++){
        digitalWrite(led,HIGH);
        delay(flash_speed);
        digitalWrite(led,LOW);
        delay(flash_speed);
        digitalWrite(led,HIGH);
    }
}

/**
 * Affichage de présentation du jeu
 */
void DisplayBegin(){
    // Print a message to the LCD.
    lcd.setCursor(3,0);
    lcd.print("MasterMind");
    lcd.setCursor(6, 1);
    lcd.print("v2.1");
    BlinkLed(10, 100);
    lcd.clear();
}

/**
 * [Affiche un écran avec la réponse et si on a gagné ou perdu]
 * @param win      [si gagné ou perdu]
 * @param secretAnswer [Réponse secrète]
 */
void DisplayEnd(int win, char *secretAnswer){
    // define if we won or loss and display a correct sentence
    if(win) {
        char *text = "CONGRATULATIONS_YOU_WIN!";
        char *text2 = "CONGRATULATIONS!";
        ScrollText(text);
        BlinkLcd(text2);
    }
    else{
        lcd.clear();
        lcd.setCursor(3,0);
        lcd.print("GAME_OVER");
    }
    // print the answer on the lcd screen
    lcd.setCursor(0,1);
    lcd.print("Answer was: ");
    lcd.write(secretAnswer);
    //delay(4000);
}

/**
 * affichage du menu de choix de niveau

```

```

 * @return [ numero du niveau ]
 */
int MenuDisplay(void){
    char *levels[5]={ "BEGINNER" , "NORMAL" , "TEACHER" , "HARD" , "42" };

    int input = 0;
    lcd.clear();
    lcd.home();
    lcd.write("SELECT_LEVEL:");
    lcd.setCursor(0,1);
    lcd.print(levels[input]);

    for(;;){
        if(GetButtons() == 1){
            lcd.clear();
            lcd.home();
            lcd.write("SELECT_LEVEL:");
            lcd.setCursor(0,1);

            input = (input+1)%5;
            lcd.print(levels[input]);
        }
        if(GetButtons() == 2){
            return input;
        }
        delay(300);
    }
}

/**
 * Fonction d'acquisition de l'état des boutons
 * On attend que l'utilisateur appuie sur un bouton et lorsqu'il le fait on retourne le
 * chiffre correspondant
 *
 * @switchState [int] est une variable qui nous sert à stocker le numero du bouton
 * allumé.
 * @return un entier qui correspond au bouton qui est allumé
 */
int GetButtons(){
    while((digitalRead(button_1) != HIGH) && (digitalRead(button_2) != HIGH)){}
    if(digitalRead(button_1) == HIGH)
        return 1;

    if(digitalRead(button_2) == HIGH)
        return 2;

    else
        return 0;
}

/**
 * Fonction de l'affichage des résultat
 *
 * Le but de cette fonction est de réaliser le caractère spécial correspondant
 * aux résultats de la réponse précédente du joueur.
 *
 * Nous commençons par créer un tableau de bytes éteint pour le cas où il n'y a aucune
 * bonnes réponses
 *
 * Ensuite nous modifions ce tableau de byte éteint en y remplaçant un élément par
 * B11000 (ligne gauche, bien placés) ou B00011 (ligne droite, mal placés) en fonction
 * des résultats
 *
 * Puis nous créons le caractère spécial désiré avec ce tableau de byte

```

```

/*
* enfin nous plaçons le curseur de l'écran lcd à la bonne position et on affiche
* le caractère créé en veillant à ce qu'il ne sorte pas de la zone d'affichage
*
*
* @param results [tableau de int] results[0] est le nombre d'éléments bien placés
* @param dx      [int *] Pointeur qui contient la position en x du curseur
*                  sur l'écran
* @param dy      [int *] Pointeur qui contient la position en y du curseur
*                  sur l'écran
* @param nbTry   [int] numéro de la tentative qui sert a definir un nouveau
*                      caractère spécial correspondant a la réponse que nous
*                      voulons afficher.
*/
void Printer(int *results, int *dx, int *dy, int nbTry) {
    /** Déclaration caractère vide */
    byte figure[8] = {B00000,B00000,B00000,B00000,B00000,B00000,B00000,B00000};

    /**
     * On va modifier ce tableau afin qu'il corresponde aux résultats s'il y a des
     * éléments bien ou mal placés
     */
    if((results[0] != 0) || (results[1]!=0)){
        /**
         * p est une variable qui va nous permettre de nous déplacer dans le tableau
         * de bytes créé précédemment
         */
        int p =0;

        /**
         * on va s'occuper de créer les lignes correspondante au nombre d'éléments
         * bien placés
         */
        for(int i = 0; i<results[0]&& p<=9; i++){
            figure[p]=B11000;
            p+=2;
        }
        /**
         * on définit p à 1 afin d'avoir le décalage entre les lignes de bien placés
         * et les lignes de mal placés
         */
        p=1;

        /**
         * Et ici on va créer les lignes correspondant au nombre d'éléments mal placés
         */
        for(int i = 0; i<results[1] && p<=9; i++){
            figure[p]=B000011;
            p+=2;
        }
    }

    /**
     * On s'occupe de créer le caractère spécial à partir du tableau figure
     * modifié (ou pas) précédemment et de réaliser l'affichage au bon endroit sur
     * l'écran LCD
     */
    lcd.createChar(nbTry+1, figure);
    lcd.setCursor(*dx,*dy);
    lcd.write(nbTry+1);

    /**
     * on veille ici a ne pas sortir de la zone d'affichage de l'écran LCD
     */
    if((*dx+1)==15){
        *dy=(*dy+1);
    }
}

```

```

        }
        *dx=(*dx+1)%15;

        delay(600);

    }

/***
 * Le calcul du résultat s'effectue en deux temps
 *
 * 1. Détermination des lettres bien placées
 * Pour cela nous comparons les lettres situées aux même positions (user[i]
 * avec secret[i]) et en cas d'égalité
 *      - on incrémente la case du tableau result correspondant au lettres
 *        bien placée
 *      - afin d'éviter un doublon dans les recherche suivantes on met la
 *        lettre du tableau secret en minuscule cela nous permettant également
 *        de ne pas perde le mot secret
 *      - toujours dans le but d'éviter une double occurrence pour les
 *        prochaine recherche on met la lettre du tableau user a la valeur 1
 *        (nous pouvons écraser cette donnée car nous ne la réutiliserons plus)
 *
 * 2. Détermination des lettres mal placées
 * Pour cela nous procédons de la même manière que précédemment mais cette fois
 * ci en comparant chaque lettre du tableau user a chaque lettre du tableau secret
 * Nous remettons ensuite les lettre du tableau secret en majuscule
 *
 * @param secretAnswer [tableau de char] contient la réponse secrète
 * @param playerAnswer [tableau de char] contient la réponse du joueur
 * @return result      [tableau d'int] tableau contenant le nombre de bien placés,
 *                     et le nombre de mal placé
 *
 */
int *GetResults(char *secretAnswer, char *playerAnswer){
    int i, j;

    /**
     * Tableau de Résultat la première(0) case correspond au éléments bien
     * placés et la deuxième(1) aux éléments mal placés
     */
    int *result = (int *)malloc(sizeof(int)*2);
    result[0] = 0; result[1] = 0;

    /**
     * Détermination des lettres bien placées
     */
    for (i = 0; i < 4; ++i)
    {
        if (playerAnswer[i] == secretAnswer[i])
        {
            result[0] += 1; /*Incrémentation du nombre de lettres bien placées*/
            secretAnswer[i] += 32; /*Mise en minuscule de la lettre*/
            playerAnswer[i] = '1'; /*Mise a 1 de la lettre*/
        }
    }
    /**
     * Si toutes les lettres sont bien placées on ne vérifie pas si il y en a des mal
     * placées
     */
    if (result[0] != 4)
    {
        for (i = 0; i < 4; i++) /*Détermination des lettres bien placées*/
        {
            for (j = 0; j < 4; j++)
            {

```

```

                if (playerAnswer[i] == secretAnswer[j])
                {
                    result[1] += 1; /*Incrémentation du nombre de lettres bien placées
                                     */
                    secretAnswer[j] += 32; /*Mise en minuscule de la lettre*/
                    playerAnswer[i] = '2'; /*Mise à 2 de la lettre*/
                }
            }
        }
    }
    for (int i = 0; i < 4; ++i) /*mise en majuscule des lettres ayant été mises en
                                 minuscule*/
    {
        if (secretAnswer[i] > 96)
        {
            secretAnswer[i] = secretAnswer[i] -= 32;
        }
    }
    return result;
}

/***
 * Fonction qui a pour objectif d'informer s'il y a déjà la nouvelle lettre dans
 * le tableau en entrée
 *
 * @param secretAnswer [tableau de char] tableau à vérifier
 * @param tmp          [char] lettre à comparer
 * @return             [int] return 1 s'il n'y a pas de jumeau, 0 sinon
 */
int CheckTwins(char *secretAnswer, char tmp){
    for (int i = 0; i < 4; i++)
        if (tmp == secretAnswer[i])
            return 0;
    return 1;
}

/***
 * Fonction qui crée la réponse secrète en fonction du niveau du jeu.
 * En effet ce dernier influe sur la plage de lettres possible pour chaque choix.
 * On utilise le code ASCII pour les manipulations.
 *
 * - On regarde tout d'abord si on a le droit d'avoir plusieurs fois la même lettre
 *   en fonction du niveau
 *   - Si on a droit à plusieurs fois la même lettre :
 *       - On choisit les 4 éléments de la réponse secrète indépendamment. Pour chacun d'
         entre eux
           on utilise une valeur aléatoire modulé par la quantité de lettres possibles
           et on additionne ça à 65 afin d'avoir la lettre correspondante.
 *   - Sinon on fait la même chose mais avant de stocker la lettre trouvée,
 *       on vérifie qu'elle n'a pas déjà été utilisée dans cette réponse
 *       et si elle l'a été on incrémente de 1 et on revérifie
 *
 * @param secretAnswer [Tableau de char contenant la réponse secrète]
 * @param interval     [Entier donnant le nombre de lettres disponibles ne fonction du
         niveau]
 * @param twins        [Entier indiquant si avoir plusieurs fois la même lettre est
         autorisé ou non]
 */
void RandomAnswer(char *secretAnswer, int interval, int twins){
    char tmp;
    ResetAnswer(secretAnswer);
    if (twins != 1){
        for (int i = 0; i < 4;){
            tmp = (rand() % interval) + 65;

```

```

        if (CheckTwins(secretAnswer , tmp) ==1){
            secretAnswer[ i ] = tmp;
            i++;
        }
    }
else
    for ( int i = 0; i < 4; i++){
        secretAnswer[ i ] = (rand()%interval)+65;
    }
secretAnswer[4]= '\0';
}

/***
 * Fonction d'acquisition de la réponse du joueur
 *
 * Nous faisons appel a la fonction GetButtons(...) pour obtenir les infor , afin de
 * récupérer l'information des boutons.
 *
 * Pour chacun des 4 éléments de sa réponse le joueur va faire défiler les
 * lettres possibles avec le bouton 1.
 *
 * A chaque fois qu'il appuie sur le bouton 1, on va afficher la lettre
 * actuellement choisie.
 *
 * Il va ensuite valider la lettre choisie avec le bouton 2. Et le curseur
 * va se déplacer d'une case sur l'écran.
 *
 * Chaque lettre choisie sera stockée à la place correspondante dans le
 * tableau "playerAnswer"
 *
 * Nous utilison la table ASCII pour les lettres. Ce qui permet une manipulation
 * plus simple. En effet pour passer a la lettre suivante il suffit d'incrémenter
 * sa valeur de 1 et de faire un modulo afin de ne pas sortir de l'ensemble de
 * lettre possible (de A à F ici en l'occurence).
 *
 * @param playerAnswer [char *] Tableau de 4 char contenant la réponse du joueur
 * @param dx           [int *] Pointeur qui contient la position en x du curseur
 *                      sur l'écran
 * @param dy           [int *] Pointeur qui contient la position en y du curseur
 *                      sur l'écran
 */
void InputLetters(char *playerAnswer , int *dx , int *dy , int interval , int twins){
    int input = 0;
    int letter = 0;
    ResetAnswer(playerAnswer);

    if(twins!=1){
        for(;letter < 4;){
            lcd.setCursor(*dx,*dy);
            lcd.write(input+65);

            if (GetButtons() ==1){
                do
                {
                    input = (input+=1)%5;

                }while (CheckTwins(playerAnswer ,(input)%interval+65)!=1);
            }
            if (GetButtons() ==2){
                playerAnswer[letter] = input + 65;
                for(input = 0;CheckTwins(playerAnswer ,input+65)!=1;input ++){}
            }
            letter++;
            *dx +=1;
        }
    }
}

```

```

        }
        delay(300);
    }
}
else{
    for(;letter < 4;){
        lcd.setCursor(*dx,*dy);
        lcd.write(input+65);

        if(GetButtons() == 1){
            input = (input+1)%interval;
        }
        if(GetButtons() == 2){
            playerAnswer[letter] = input + 65;
            input = 0;
            letter++;
            *dx +=1;
        }
        delay(200);
    }
}

/***
 * Boucle de jeu principale.
 *
 * elle depend directement du choix de niveau effectué dans le menu.
 *
 * en fonction du niveau choisi elle va autoriser ou non d'avoir plusieurs
 * fois la même lettres et la quantité de lettre disponible
 *
 * @param secretAnswer [réponse secrète]
 * @param playerAnswer [réponse du joueur]
 * @param dx           [position du curseur sur l'écran en x]
 * @param dy           [position du curseur sur l'écran en y]
 * @param interval    [quantité de lettres disponible]
 * @param twins        [si avoir plusieurs fois la même lettre est autorisé ou non]
 * @return             [0 si victoire , 1 sinon]
 */
int GameLoop(char *secretAnswer, char *playerAnswer, int dx, int dy, int interval, int
twins){
    int nbTry;
    int *result;

    RandomAnswer(secretAnswer, interval, twins);

    for(nbTry = 0; nbTry<6; nbTry++){
        InputLetters(playerAnswer, &dx, &dy, interval, twins);
        result = GetResults(secretAnswer, playerAnswer);
        Printer(result, &dx,&dy, nbTry);

        if(result[0] == 4)
            return 0;
    }
    return 1;
}

/***
 * Boucle principale du programme
 *
 * Dans cette version nous avons implémenté des niveaux, il y en a 5:
 * - Beginner : niveau débutant, "finger in the nose"
 * - Pas le droit à plusieurs fois la même lettre
 * - Les lettres possible sont de A à E
 */

```

```

/*
*   - Normal : règles officielle du Mastermind
*       - Pas le droit à plusieurs fois la même lettre
*       - Les lettres possible sont de A à F
*
*   - Teacher : règles demandées dans le sujet du projet
*       - Plusieurs même lettres possible
*       - Les lettres possible sont de A à F
*
*   - Hard : niveau difficile , "Good Luck !"
*       - Plusieurs même lettres possible
*       - Les lettres possible sont de A à G
*
*   - 42 : niveau solution 404 "Fuyez Pauvres Fous!"
*       - Plusieurs même lettres possible
*       - Les lettres possible sont tout l'alphabet
*
*
* Et nous avons également des Leds qui s'allument en fonction des évenement du jeu.
;)
*/
* Enjoy the game !
*/
void loop() {
    char secretAnswer[4];
    char playerAnswer[4];
    int dx = 0;
    int dy = 0;
    int gameStatus = 0;
    int level = 4;

    lcd.clear();

    level = MenuDisplay();

    lcd.clear();
    delay(300);

    switch(level){
        case 0:
            gameStatus = GameLoop(secretAnswer, playerAnswer, dx, dy, 5, 0);
            break;

        case 1:
            gameStatus = GameLoop(secretAnswer, playerAnswer, dx, dy, 6, 0);
            break;

        case 2:
            gameStatus = GameLoop(secretAnswer, playerAnswer, dx, dy, 6, 1);
            break;

        case 3:
            gameStatus = GameLoop(secretAnswer, playerAnswer, dx, dy, 7, 1);
            break;

        case 4:
            gameStatus = GameLoop(secretAnswer, playerAnswer, dx, dy, 26, 1);
            break;
    }

    if(gameStatus == 0){
        DisplayEnd(1, secretAnswer);
        LedAnswer(led_v,100,10);
        delay(1000);
    }
}

```

```
}

else{
    DisplayEnd(0, secretAnswer);
    LedAnswer(led_r,100,5);
    delay(1000);

}
delay(1000);

digitalWrite(led_v,LOW);
digitalWrite(led_r,LOW);

}
```

## 6.2 Quelques images de la version finale :

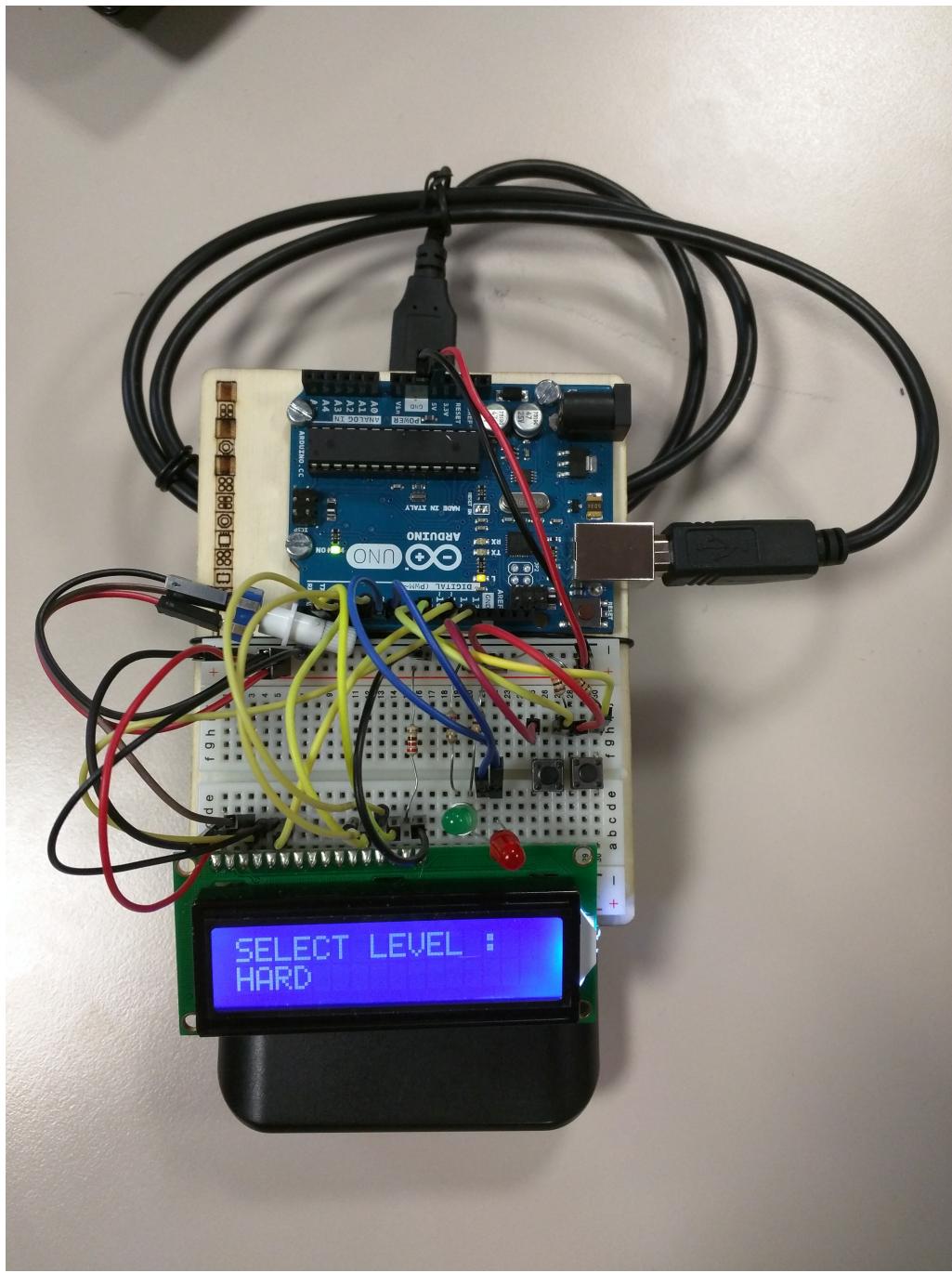


FIGURE 6.1 – Montage final avec batterie pour pouvoir jouer partout

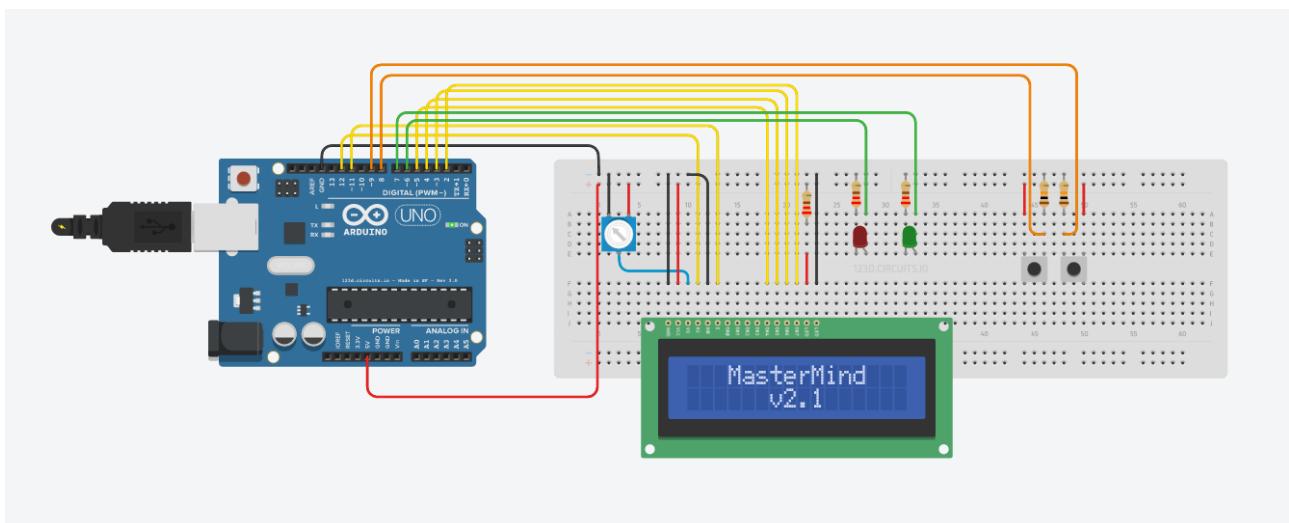
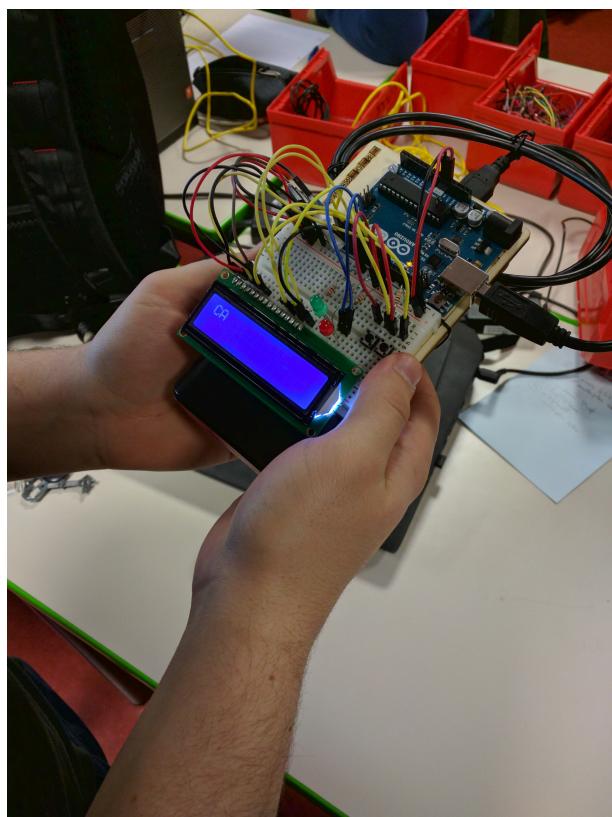
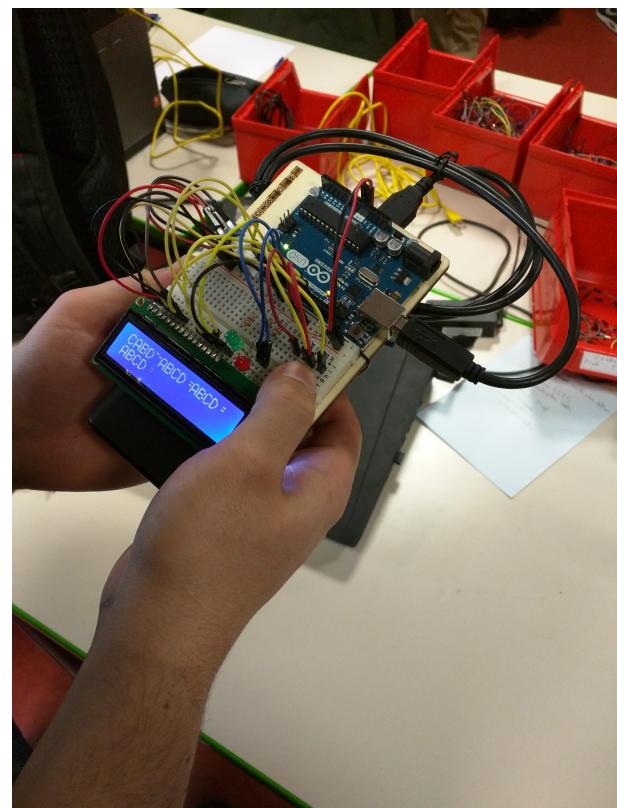


FIGURE 6.2 – Montage final du Projet Mastermind



(a) Début de partie



(b) Partie de Mastermind

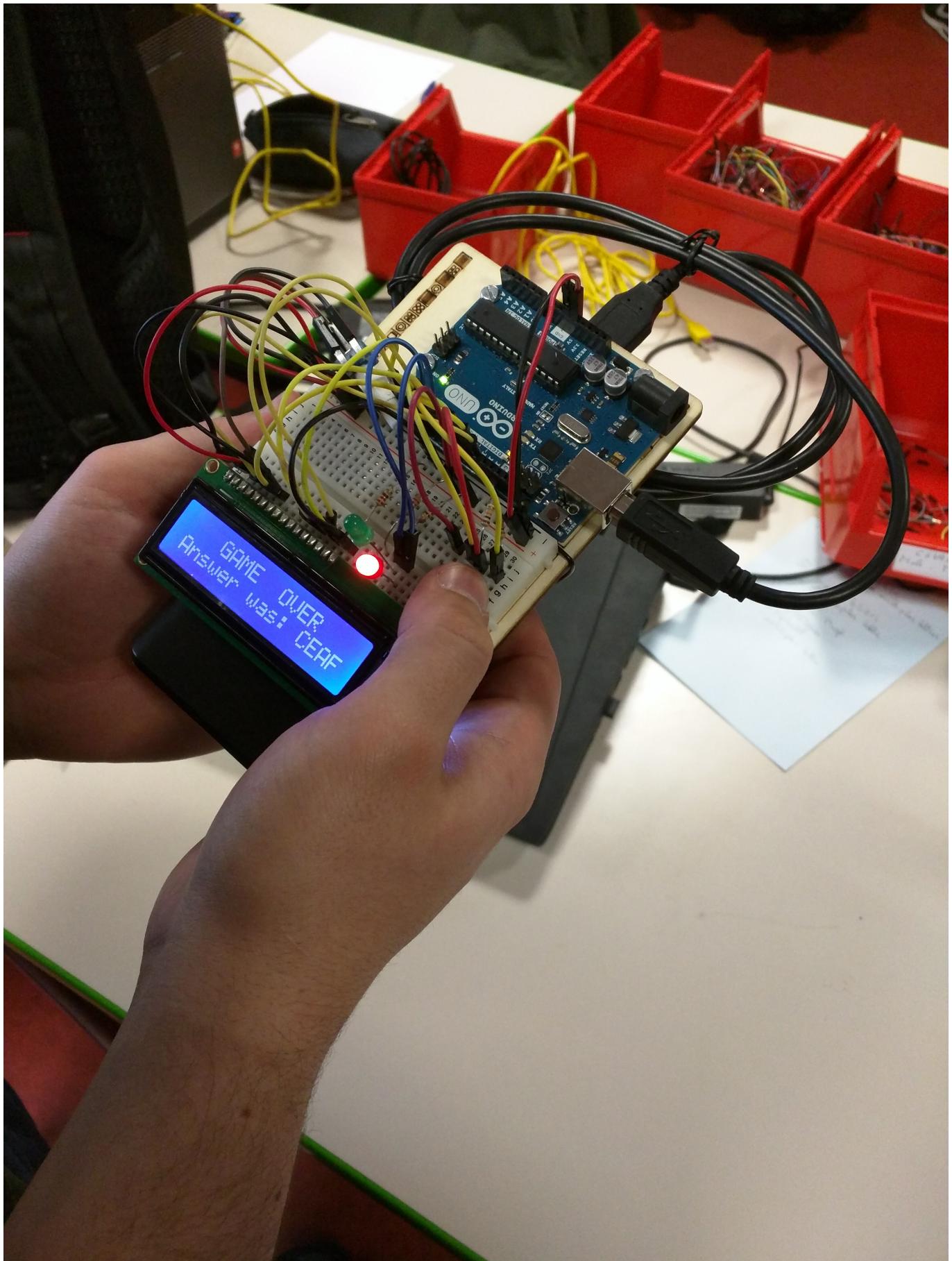


FIGURE 6.3 – Fin de partie