Montage

12 Le clavier numérique

Au sommaire :

- la fabrication d'un clavier numérique ;
- apprendre à interroger les différentes touches élégamment ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire.

Qu'est-ce qu'un clavier numérique ?

Vous connaissez déjà le bouton-poussoir, dont nous avons parlé au cours de certains montages. Mais ici, plusieurs boutons-poussoirs (touches) sont réunis en une matrice – donc disposés en lignes et colonnes – de manière à proposer les chiffres 0 à 9 et deux touches spéciales telles que * et #. À quoi cela sert-il ? Eh bien, vous utilisez ce jeu de touches tous les jours, entre autres pour téléphoner.

◀ **Figure 12-1**
Clavier de téléphone

Il s'agit d'une matrice de 4 × 3 touches (4 lignes et 3 colonnes). Cette matrice est également appelée *keypad* (clavier numérique) et peut être achetée prête à l'emploi en différentes variantes. La figure 12-2 montre deux claviers numériques à film. Celui de gauche possède même quelques touches supplémentaires A à D, qui peuvent s'avérer très utiles si les 12 touches du clavier numérique de droite ne suffisent pas pour votre projet.

Figure 12-2 ▶
Clavier numérique à film 4 × 4 à 16 touches et clavier numérique à film 4 × 3 à 12 touches.

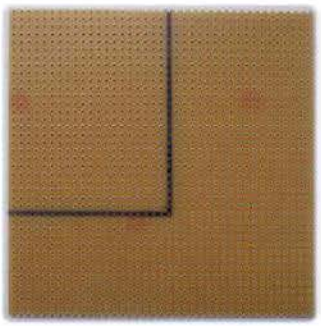
Je vois mal comment brancher par exemple le clavier numérique à film 4 × 4 sur mon Arduino sans rencontrer des problèmes de broches. On pourrait bien sûr raccorder les 16 touches d'un côté aux +5 V et les 16 prises correspondantes aux entrées numériques. On pourrait également se servir des entrées analogiques en cas de besoin. C'est ce que nous avons fait déjà.

Vous pourriez bien sûr procéder ainsi et cela fonctionnerait s'il n'y avait pas les limites physiques de la carte Arduino Uno. Une solution consisterait à utiliser la carte Arduino Mega, dont le nombre d'interfaces est bien élevé. Mais soyons ingénieux ; il existe une bibliothèque pour claviers numériques prête à l'emploi sur le site Internet Arduino, aussi allons-nous tout faire par nous-mêmes. Nous utiliserons le clavier numérique 4 × 3 que nous aurons fabriqué de nos propres mains. Voici la liste du matériel nécessaire.

Composants nécessaires

12 boutons-poussoir

1 jeu de connecteurs femelles empilables



1 carte de dimensions 10 × 10 ou mieux 16 × 10 (vous pourrez alors en faire deux shields). La découpe du shield est déjà indiquée, et je reviendrai bientôt sur cette dernière.

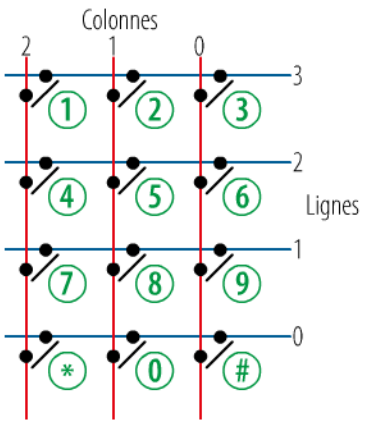
Fil, si possible de différentes couleurs

Réflexions préliminaires

Ardus vient de nous faire remarquer que notre clavier numérique 4 × 4 nécessitait 16 lignes pour interroger toutes les touches. Un clavier numérique 4 × 3 n'aurait quant à lui besoin que de 12 lignes. Mais ce serait encore beaucoup trop à mon avis. Une solution astucieuse existe, dont l'idée de base a déjà servi pour commander les deux afficheurs sept segments. Vous vous demandez sûrement ce que des afficheurs sept segments ont à voir avec ces touches. Le mot commun est multiplexage. Il signifie que certains signaux sont regroupés et envoyés par un moyen de transmission pour minimiser l'utilisation des lignes et en tirer profit le plus possible. Sur les afficheurs sept segments, les lignes de commande de deux segments sont montées en parallèle et utilisées pour commander les deux. Sept ou huit lignes par segment sont ainsi économisées. La solution trouvée

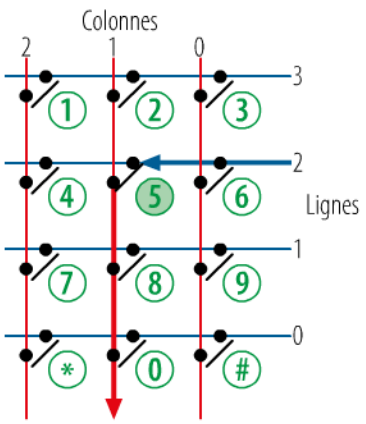
pour interroger les différentes touches d'un clavier numérique est relativement simple. Voici le câblage des 12 touches.

Figure 12-3 ▶
Câblage des 12 touches d'un clavier numérique 4 × 3



Imaginez une grille composée de 4 + 3 fils métalliques posés l'un sur l'autre sans pour autant se toucher. Voilà à quoi ressemble ce graphique. On voit que les 4 fils horizontaux en bleu forment des lignes numérotées de 0 à 3. Au-dessus, les trois fils verticaux en rouge forment à peu de distance des colonnes numérotées de 0 à 2. Chaque intersection présente des petits contacts reliant, quand on appuie sur la touche, la ligne et la colonne en question pour former un tronçon de circuit électrique. Regardez bien la figure 12-4, où la touche 5 est enfoncée.

Figure 12-4 ▶
La touche 5 est enfoncée (les lignes en gras montrent le passage du courant).



Le courant peut alors passer de la ligne 2 via l'intersection 5 dans la colonne 1 et y être détecté.

Mais si une tension est appliquée simultanément à toutes les lignes, la touche 2 au-dessus de la touche 5 peut tout aussi bien être appuyée et j'enregistrerai une impulsion correspondante sur la colonne 1. Comment faire la différence ?

Je vois, Ardus, que vous n'avez pas complètement compris le principe. Pas de quoi fouetter un chat cependant. Disons grossièrement que nous envoyons tour à tour un signal par les lignes 0 à 3 et interrogeons ensuite également tour à tour le niveau sur les colonnes 0 à 2. Le déroulement est alors le suivant :

- **Niveau `HIGH` sur le fil de la rangée 0**
  - Interrogation du niveau sur la colonne 0
  - Interrogation du niveau sur la colonne 1
  - Interrogation du niveau sur la colonne 2
- **Niveau `HIGH` sur le fil de la rangée 1**
  - Interrogation du niveau sur la colonne 0
  - Interrogation du niveau sur la colonne 1
  - Interrogation du niveau sur la colonne 2
  - etc.

Cette interrogation est bien sûr si rapide que suffisamment de passages ont lieu en une seule seconde pour que pas un appui de touche ne soit omis. Le shield a été câblé à demeure avec les numéros de broches des entrées et sorties indiqués sur la figure 12-5.

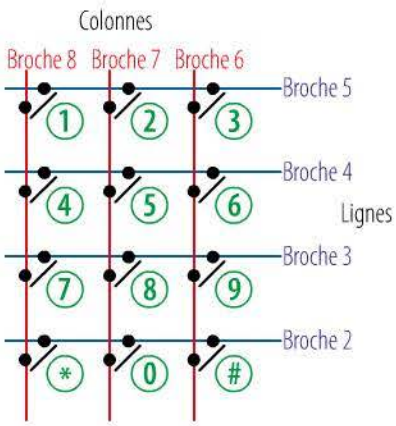


Figure 12-5
Câblage des différentes lignes et colonnes avec les broches numériques

Pimentons ici un peu les choses et créons notre propre bibliothèque qui servira plus tard à d'autres montages. Elle offre une certaine fonctionnalité de base et pourra bien entendu être modifiée ou élargie si

besoin est. Le sketch principal demande continuellement au shield quelle touche a été appuyée. Le résultat est affiché dans le Serial Monitor pour visualisation. Pour vérifier le bon fonctionnement de la bibliothèque, fixons-nous les spécifications suivantes :

- si vous n'appuyez sur aucune touche, aucun caractère n'est affiché dans le Serial Monitor ;
- si vous n'appuyez que brièvement sur une touche, le chiffre ou le caractère s'affiche dans le moniteur ;
- si vous appuyez sur une touche un long moment, qui peut être préalablement défini en conséquence, le chiffre ou le caractère s'affiche plusieurs fois l'un derrière l'autre jusqu'à ce que la touche soit relâchée.

Code du sketch

Sketch principal avec revue de code

Commençons par le sketch principal qui, du fait de la fonctionnalité délocalisée dans une bibliothèque, semble clair pour ne pas dire spartiate. Mais attendez seulement. Les choses vont devenir plus complexes et plus intéressantes.

```
#include "MyKeyPad.h"
int rowArray[] = {2, 3, 4, 5}; //Initialiser le tableau avec les
                               //numéros de broche des lignes
int colArray[] = {6, 7, 8};    //Initialiser le tableau avec les
                               //numéros de broche des colonnes
MyKeyPad myOwnKeyPad(rowArray, colArray);//Instanciation d'un objet
void setup(){
  Serial.begin(9600);                 //Préparer la sortie série
  myOwnKeyPad.setDebounceTime(500);   //Régler le temps du
                                      //rebond à 500 ms
}
void loop(){
  char myKey = myOwnKeyPad.readKey(); //Lecture de la touche appuyée
  if(myKey != KEY_NOT_PRESSED)        //Une touche quelconque
                                      //a-t-elle été appuyée ?
     Serial.println(myKey);           //Impression du caractère
                                      //de la touche
}
```

La première ligne incorpore, tout comme dans la bibliothèque-dé du montage n° 9, le fichier d'en-tête permettant d'utiliser la bibliothèque. Nous verrons bientôt ce qu'il contient. Déclarons pour

commencer deux tableaux, que nous initialisons avec les numéros des broches de connexion aux lignes et aux colonnes du clavier numérique. Cela offre une plus grande flexibilité et permet d'adapter les constructions différentes. La ligne :

```
MyKeyPad myOwnKeyPad(rowArray, colArray);
```

génère l'instance `myOwnKeyPad` de la classe `MyKeyPad` qui est définie dans la bibliothèque, et transmet les deux tableaux au constructeur de la classe. Ces informations lui sont nécessaires pour commencer à évaluer sur laquelle des 12 touches on a appuyé. Le temps de rebond est déterminé par la ligne suivante :

```
myOwnKeyPad.setDebounceTime(500);
```

La méthode `setDebounceTime` avec l'argument `500` est ainsi appelée. L'instance est ensuite continuellement interrogée au sein de la fonction `loop`, la question posée étant : « Indique moi la touche qui est actuellement appuyée sur le clavier numérique ! » Pour y arriver, il faut écrire la ligne suivante :

```
char myKey = myOwnKeyPad.readKey();
```

Elle affecte le résultat de la requête à la variable `myKey` du type `char`. On peut maintenant réagir en conséquence. Il le faut car la méthode renvoie toujours une valeur, qu'une touche ait été appuyée ou non. Mais vous souhaitez sûrement voir à l'écran si une touche a été appuyée. Aussi la valeur `KEY_NOT_PRESSED` est-elle renvoyée quand aucune touche n'est appuyée. La requête `if` n'envoie donc le caractère correspondant à la touche au Serial Monitor que si une touche est véritablement appuyée.

```
if(myKey != KEY_NOT_PRESSED)
    Serial.println(myKey);
```

Une question en passant : qu'y a-t-il derrière `KEY_NOT_PRESSED` ?

Question pertinente car j'en serais venu de toute façon à parler du fichier d'en-tête. De nombreuses constantes symboliques y sont définies. Parmi ces constantes se cache le caractère -, qui est toujours envoyé lorsqu'aucune touche n'est appuyée. Je lui ai donné ce nom évocateur pour que le code soit plus lisible.

Fichier d'en-tête avec revue de code

Le fichier d'en-tête sert, comme nous l'avons déjà expliqué, à faire connaître les variables et les méthodes nécessaires à la définition de la classe en question. Voyons maintenant ce qu'on y trouve :

```
#ifndef MYKEYPAD_H
#define MYKEYPAD_H

#if ARDUINO < 100
#include <WProgram.h>
#else
#include <Arduino.h>
#endif

#define KEY_NOT_PRESSED '-'  //Nécessaire si aucune touche
                             //n'est appuyée
#define KEY_1 '1'
#define KEY_2 '2'
#define KEY_3 '3'
#define KEY_4 '4'
#define KEY_5 '5'
#define KEY_6 '6'
#define KEY_7 '7'
#define KEY_8 '8'
#define KEY_9 '9'
#define KEY_0 '0'
#define KEY_STAR '*'
#define KEY_HASH '#'

class MyKeyPad{
  public:
    MyKeyPad(int rowArray[], int colArray[]); //Constructeur
                                              //paramétré
    void setDebounceTime(unsigned int debounceTime);
                        //Réglage du temps de rebond
    char readKey();     //Détermine la touche appuyée sur
                        //le clavier numérique
  private:
    unsigned int debounceTime; //Variable locale pour temps de rebond
    long lastValue; //Dernière valeur de la fonction millis
    int row[4];     //Tableau pour les lignes
    int col[3];     //Tableau pour les colonnes
};
#endif
```

La partie supérieure est consacrée aux constantes symboliques et aux caractères correspondants. Vient ensuite la définition formelle de la

classe sans formulation du code qui, comme chacun sait, se trouve dans le fichier .cpp.

Eh, une minute ! Vous cherchez encore à me faire gober quelque chose que je ne connais pas. Que veut dire `unsigned int` dans la déclaration des variables ?

Eh bien Ardus, vous avez une bien piètre opinion de moi ! Évidemment que j'allais en parler ! Le type de donnée `int` vous est déjà familier. Son domaine s'étend des valeurs négatives aux valeurs positives. Le mot-clé `unsigned` placé devant indique que la variable est déclarée sans signe, autrement dit son domaine de valeurs double puisque les valeurs négatives sont supprimées. Ce type de donnée nécessite également (comme `int`) de deux octets pour que les valeurs positives soient toutes représentées. Le domaine de valeurs va de 0 à 65 535.

Fichier ccp avec revue de code

Voici maintenant un peu de code « fait maison » :

```
#include "MyKeyPad.h"
//Constructeur paramétré
MyKeyPad::MyKeyPad(int rowArray[], int colArray[]){
  //Copier le tableau des broches
  for(int r = 0; r < 4; r++)
    row[r] = rowArray[r];
  for(int c = 0; c < 3; c++)
    col[c] = colArray[c];
  //Programmation des broches numériques
  for(int r = 0; r < 4; r++)
    pinMode(row[r], OUTPUT);
  for(int c = 0; c < 3; c++)
     pinMode(col[c], INPUT);
  //Définition initiale de debounceTime à 300 ms
  debounceTime = 300;
}
  //Méthode pour régler le temps de rebond
  void MyKeyPad::setDebounceTime(unsigned int time){
    debounceTime = time;
  }
  //Méthode pour déterminer la touche appuyée
  //sur le clavier numérique
  char MyKeyPad::readKey(){
    char key = KEY_NOT_PRESSED;
    for(int r = 0; r < 4; r++){
      digitalWrite(row[r], HIGH);
```

```
      for(int c = 0; c < 3; c++){
        if((digitalRead(col[c]) == HIGH)&&(millis() - lastValue) >=
                                          debounceTime){
          if((c==2)&&(r==3)) key = KEY_1;
          if((c==1)&&(r==3)) key = KEY_2;
          if((c==0)&&(r==3)) key = KEY_3;
          if((c==2)&&(r==2)) key = KEY_4;
          if((c==1)&&(r==2)) key = KEY_5;
          if((c==0)&&(r==2)) key = KEY_6;
          if((c==2)&&(r==1)) key = KEY_7;
          if((c==1)&&(r==1)) key = KEY_8;
          if((c==0)&&(r==1)) key = KEY_9;
          if((c==2)&&(r==0)) key = KEY_STAR; // *
          if((c==1)&&(r==0)) key = KEY_0;
          if((c==0)&&(r==0)) key = KEY_HASH; // #
          lastValue = millis();
        }
      }
    digitalWrite(row[r], LOW); //Restauration du niveau initial
    }
return key;
}
```

Voyons d'abord le constructeur. Il sert à initialiser l'objet à créer et à lui donner des valeurs initiales définies. Un constructeur doit permettre d'initialiser autant que possible complètement l'instance, de telle sorte qu'aucun appel de méthode ne soit plus en principe nécessaire pour l'initialisation. Elles ne sont plus utilisées que pour corriger certains paramètres qui, le cas échéant, doivent ou peuvent être modifiés en cours de sketch. Le constructeur n'est appelé qu'une seule fois et de manière implicite lors de l'instanciation, et après cela plus jamais dans la vie de l'objet. Dans notre exemple, les tableaux des lignes et des colonnes lui sont communiqués lors de l'appel, de manière à pouvoir être transmis ensuite aux tableaux locaux au moyen de deux boucles `for` :

```
//Copie des tableaux de broches
  for(int r = 0; r < 4; r++)
    row[r] = rowArray[r];
  for(int c = 0; c < 3; c++)
    col[c] = colArray[c];
```

Les broches numériques sont ensuite initialisées et leurs sens de transfert sont définis :

```
//Programmation des broches numériques
  for(int r = 0; r < 4; r++)
```

```
    pinMode(row[r], OUTPUT);
  for(int c = 0; c < 3; c++)
    pinMode(col[c], INPUT);
//Définition initiale de debounceTime à 300 ms
debounceTime = 300;
```

> Vous avez dit qu'un objet devait toujours être complètement instancié au moyen d'un constructeur. Mais vous lui transmettez uniquement les tableaux de broches pour les lignes et les colonnes. Un autre paramètre important est néanmoins le temps de rebond. Celui-ci n'est pourtant pas transmis à l'objet par le constructeur. Vous avez pour ce faire une méthode propre. Cela ne contredit-il pas ce que vous venez de dire ?

Oui et non, Ardus ! Il est vrai que le constructeur ne connaît pas le temps de rebond. Mais regardez sa dernière ligne. Le temps y est réglé sur 300 ms. Il s'agit pratiquement d'une initialisation câblée en dur, comme on dit si bien dans les milieux de la programmation. Si la valeur ne vous dit rien, vous pouvez toujours l'adapter à vos besoins, tout comme je l'ai fait d'ailleurs pour la méthode `setDebounceTime`. La valeur de `300` (ms) m'a semblé ici convenir. J'aurais évidemment pu la définir directement, mais je voulais vous montrer cette possibilité. La tâche en question est accomplie par la méthode `readKey`, qui est appelée sans cesse dans la boucle `loop` pour pourvoir réagir immédiatement à un appui sur une touche. Au début de l'appel de la méthode, la ligne suivante fait en sorte que la variable `key` soit immédiatement pourvue d'une valeur initiale :

```
char key = KEY_NOT_PRESSED;
```

Allez voir dans le fichier d'en-tête de quelle valeur il s'agit.

Si aucune touche n'est en effet appuyée, c'est précisément ce signe qui est réexpédié comme résultat. Vient ensuite l'appel des deux boucles `for` imbriquées l'une dans l'autre. La première ligne du clavier numérique est mise au niveau `HIGH` par :

```
digitalWrite(row[r], HIGH);
```

Les niveaux de toutes les colonnes sont ensuite testés.

```
…
for(int c = 0 ; c < 3 ; c++){
  if((digitalRead(col[c]) == HIGH)&&(millis() - lastValue) >=
                                    debounceTime){
     if((c==2)&&(r==3)) key = KEY_1;
     if((c==1)&&(r==3)) key = KEY_2;
     if((c==0)&&(r==3)) key = KEY_3;
```

```
      if((c==2)&&(r==2)) key = KEY_4;
      if((c==1)&&(r==2)) key = KEY_5;
      if((c==0)&&(r==2)) key = KEY_6;
      if((c==2)&&(r==1)) key = KEY_7;
      if((c==1)&&(r==1)) key = KEY_8;
      if((c==0)&&(r==1)) key = KEY_9;
      if((c==2)&&(r==0)) key = KEY_STAR; //*
      if((c==1)&&(r==0)) key = KEY_0;
      if((c==0)&&(r==0)) key = KEY_HASH; //#
      lastValue = millis();
    }
  }
…
```

Si une colonne présente également un niveau `HIGH` et si en plus le temps de rebond a été pris en compte, alors la première condition `if` est remplie et toutes les conditions `if` subséquentes sont évaluées. Si une condition est vérifiée pour le compteur de lignes `r` et le compteur de colonnes `c`, la variable `key` est initialisée avec la valeur initiale correspondante et renvoyée à l'appelant, en fin de méthode, par l'instruction `return`. Une fois la boucle intérieure terminée, les lignes qui viennent d'être mises au niveau `HIGH` doivent être remises dans leur état initial, qui est le niveau `LOW`. Si l'état `HIGH` était conservé, une interrogation ciblée d'une certaine ligne ne serait alors plus possible. Toutes les lignes auraient un niveau `HIGH` une fois la boucle extérieure terminée, ce qui mettrait toute la logique d'interrogation sens dessus dessous.

Je ne comprends pas bien ce qui se passe avec le *temps de rebond*. Réexpliquez moi la fonction en question s'il vous plaît. J'ai bien compris qu'elle était nécessaire mais je ne vois pas bien comment tout cela fonctionne.

Mais volontiers, Ardus ! La fonction `millis` renvoie le nombre de millisecondes écoulées depuis le début du sketch. La dernière valeur est pour ainsi dire stockée temporairement dans la variable `lastValue` une fois la boucle intérieure terminée. Si la boucle est de nouveau appelée, la différence entre la valeur actuelle en millisecondes et la valeur précédente est calculée. Ce n'est que si elle est supérieure au temps de rebond défini que la condition est jugée *vérifiée*. Elle se trouve cependant liée avec l'expression qui la précède par un `&` logique.

```
if((digitalRead(col[c]) == HIGH)&&(millis() - lastValue) >=
                                  debounceTime)…
```

Ce n'est que si les deux conditions délivrent le résultat logique *vrai* à l'instruction `if` que la ligne se poursuit avec l'accolade. Cette structure permet d'obtenir une interruption temporelle, qui se produit aussi toute seule dans certains sketches.

Schéma

Le circuit est assez simple, mais les besoins en programmation sont eux plus importants.

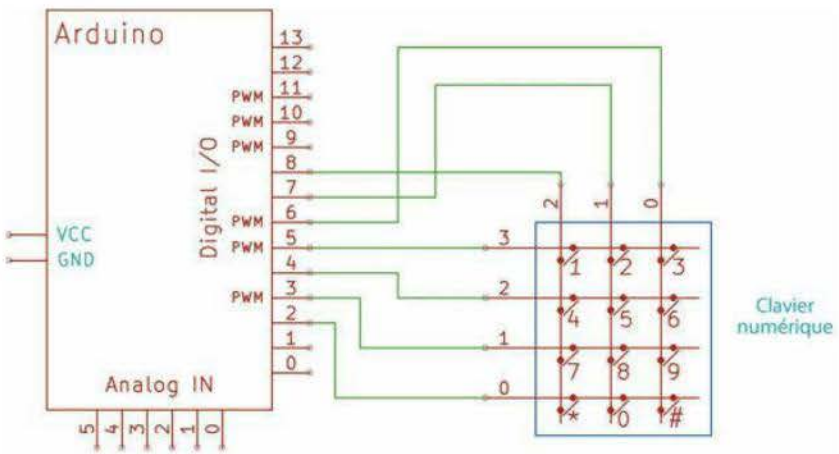


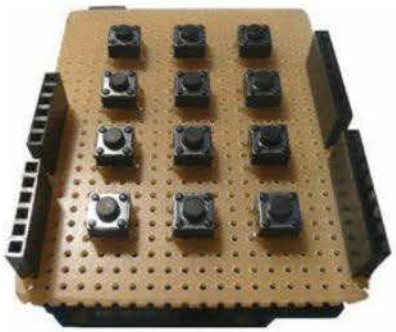
Figure 12-6
Commande de notre clavier numérique

Une chose me frappe d'emblée dans ce schéma. Si aucune touche n'est actionnée, les entrées numériques `6`, `7` et `8` se retrouvent pour ainsi dire le bec dans l'eau. N'avez-vous pas dit au début qu'une entrée devait toujours avoir un niveau défini ?

C'est vrai, Ardus ! Mais le clavier numérique doit rester relativement simple et, à moins que la foudre ne tombe sur votre siège et ne provoque une grande perturbation électrostatique de votre environnement, il fonctionne parfaitement bien. Je n'ai pas eu de problème avec ce circuit. Essayez-le seulement. Et tant que vous y êtes, réécrivez donc votre sketch de manière à utiliser les résistances pull-up internes. Le shield n'a pas besoin d'être modifié. Seul le code doit être un peu adapté. Voici un indice pour commencer : si les résistances pull-up sont activées, il vous faut interroger les différentes broches non pas sur leur niveau `HIGH` mais sur leur niveau `LOW`. À vous de trouver le reste par vous-même. Considérez cela comme une partie de l'exercice à venir.

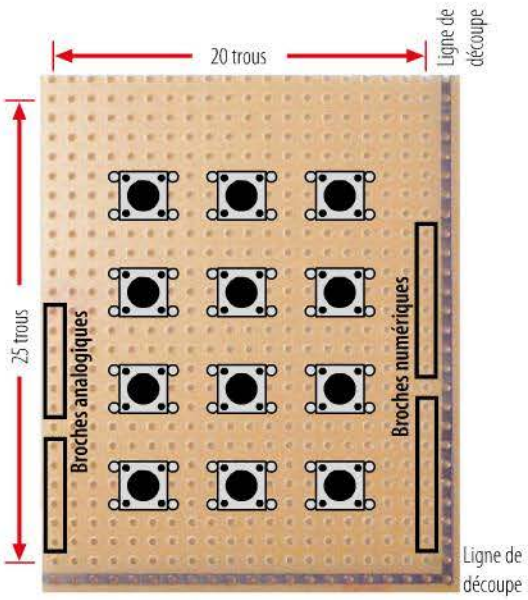
Réalisation du shield

Figure 12-7 ▶
Réalisation du clavier numérique avec son propre shield



La construction du shield n'est pas mal du tout, n'est-ce pas ? Je vous avais dit au début que je vous montrerai comment faire une carte à la bonne taille. La carte présentée page 399 comporte un marquage correspondant à la taille définitive du shield. Vous trouverez des informations détaillées sur sa fabrication dans le montage n° 22 de réalisation d'un shield.

Figure 12-8 ▶
Taille du shield basée sur les écarts entre les trous



On voit sur l'image les positions exactes des connecteurs femelles et des touches. Il suffit de compter les trous sur la carte et de positionner ensuite les composants. Ne commencez à souder que quand vous avez tout placé sur la carte. Vous évitez ainsi les positionnements incorrects, et les erreurs vous sautent tout de suite aux yeux. Si vous soudez les composants aussitôt après les avoir placés, il se peut que

vous vous aperceviez plus tard que vous avez commis une erreur, et que vous ayez tout à dessouder. La figure 12-9 illustre l'envers de la carte une fois tous les composants soudés et tous les fils raccordés.

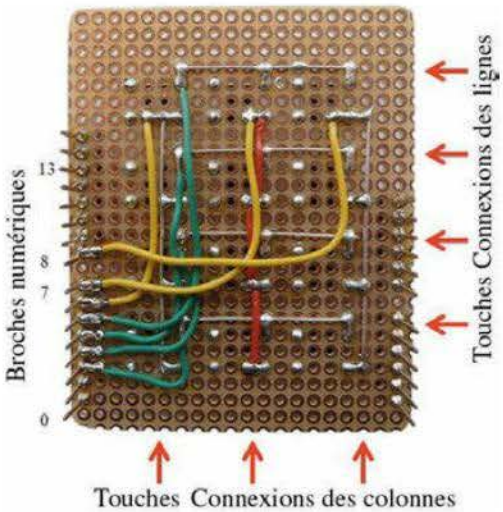


Figure 12-9
Envers de la carte

Les fils verts établissent les liaisons vers les lignes, et les fils jaunes vers les colonnes. Les fils rouges sont les connexions intermédiaires des colonnes, qui passent au-dessus des fils horizontaux.

Problèmes courants

La réalisation de ce shield nécessitant beaucoup de soudure, les erreurs peuvent être d'autant plus nombreuses.

- Vérifiez que les fils sont bien raccordés aux bonnes broches.
- Avez-vous correctement relié les différentes touches entre elles, de manière à ce qu'elles forment des lignes et des colonnes ? Servez-vous du schéma. Voyez s'il n'y a pas de court-circuit entre eux. Le mieux est de prendre une loupe et de vérifier chaque soudure. Un court-circuit de la taille d'un cheveu n'est souvent pas visible à l'œil nu.

Qu'avez-vous appris ?

- Vos avez vu qu'on peut fabriquer soi-même un clavier numérique avec des composants très simples et peu onéreux. Pour ceux qui ont la patience nécessaire et qui ont envie de faire par

eux-mêmes au lieu de toujours se servir des composants tout prêts vendus dans les magasins, ceci a pu être un bon début et leur a permis de montrer ou plutôt d'entretenir leur créativité.

- Le soudage qui, il y a des décennies, était excessivement pratiqué dans les premiers bricolages électroniques, n'est selon moi plus à la mode aujourd'hui. Mais j'espère du moins que l'odeur d'étain fondu et de plastique brûlé vous aura charmé, comme elle a su le faire dans ma jeunesse.
- Nous avons créé ensemble notre propre classe, qui peut servir à interroger la matrice de touches. Vous avez certainement tiré parti des principes de la POO, qui vous avaient été expliqués auparavant.

Exercice complémentaire

La bibliothèque `KeyPad` fait pour le moment partie du sketch que vous avez créé. Je pense que ce serait une bonne idée de la copier quelque part, à l'attention de tous les autres sketches qui pourraient en avoir besoin. Si vous ne savez plus où, relisez le montage n° 8 du dé électronique, au cours duquel vous aviez créé votre première bibliothèque. Vous y trouverez les informations nécessaires. Il faut pour ce faire rajouter le fichier `keyword.txt` dans votre bibliothèque. Entrez-y les mots-clés nécessaires, qui sont indiqués en couleurs dans l'IDE Arduino.