Montage 8

Le dé électronique

Au sommaire :

- la déclaration et l'initialisation d'un tableau bidimensionnel ;
- la programmation de plusieurs broches comme sortie (`OUTPUT`) ;
- la programmation d'un port comme entrée (`INPUT`) ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire ;
- quelque chose d'intéressant.

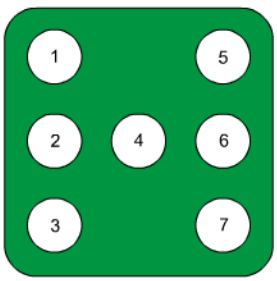
Qu'est-ce qu'un dé électronique ?

Bien que les derniers montages vous aient donné quelques bases pour programmer la carte Arduino, vous pensez sûrement être encore loin du compte… Aussi allons-nous appliquer, approfondir et élargir nos connaissances en étudiant quelques circuits intéressants. La construction d'un dé électronique est toujours plaisante. Il y a quelques années de cela, quand les microprocesseurs n'existaient pas ou étaient hors de prix, on utilisait plusieurs circuits intégrés. Vous trouverez pour ce faire de nombreuses instructions de bricolage sur Internet. Notre but est ici de commander le dé électronique avec la seule carte Arduino.

Tout le monde a déjà joué aux dés, que ce soit au 421, au 5000 ou au Yams. Aussi notre prochain circuit sera celui d'un dé électronique. Il se compose d'une unité d'affichage avec sept LED et un bouton-

poussoir pour lancer le dé. Voici d'abord la disposition des LED qui est celle des points d'un véritable dé. Les points portent tous un numéro pour mieux se repérer au moment de commander les LED. Le numéro 1 se trouve en haut à gauche, la numérotation se poursuivant vers le bas puis vers la droite jusqu'au numéro 7 qui se trouve en bas à droite.

Figure 8-1 ▶
Numérotation des points du dé



Notre construction doit comporter un bouton-poussoir qui lance le dé. Lorsqu'on appuie dessus toutes les LED clignotent irrégulièrement et quand on le relâche, l'affichage s'arrête sur une certaine combinaison de LED, laquelle représente le nombre obtenu. Les différentes combinaisons de points sont répertoriées dans le tableau 8-1.

Tableau 8-1 ▶
Quelle LED s'allume pour quel nombre ?

Dé	Nombre	LED						
		1	2	3	4	5	6	7
	1				✓			
	2	✓						✓
	3	✓			✓			✓
	4	✓		✓		✓		✓
	5	✓		✓	✓	✓		✓

Dé	Nombre	LED						
		1	2	3	4	5	6	7
	6	✓	✓	✓		✓	✓	✓

◀ Tableau 8-1 (suite) Quelle LED s'allume pour quel nombre ?

Il est certes tout à fait possible de construire le circuit sur votre plaque d'essais, mais ce n'est pas toujours simple compte tenu de la symétrie des LED. Dans un montage à part (le n° 22), nous construirons le circuit sur une carte spéciale appelée *shield*, et que nous brancherons par-dessus la carte Arduino. C'est la manière la plus propre et la plus pratique de fabriquer un dé électronique durable. Mais utilisons d'abord la plaque d'essais. Quel matériel nous faut-il ?

Composants nécessaires

	7 LED rouges
	7 résistances de 330 Ω
	1 résistance de 10 kΩ
	1 bouton-poussoir
	Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

Voici le code du sketch pour commander le dé électronique :

```
#define WAITTIME 20
int pips[6][7] = {{0, 0, 0, 1, 0, 0, 0},   //Nombre sorti 1
                  {1, 0, 0, 0, 0, 0, 1},   //Nombre sorti 2
                  {1, 0, 0, 1, 0, 0, 1},   //Nombre sorti 3
                  {1, 0, 1, 0, 1, 0, 1},   //Nombre sorti 4
                  {1, 0, 1, 1, 1, 0, 1},   //Nombre sorti 5
                  {1, 1, 1, 0, 1, 1, 1}}; //Nombre sorti 6
```

```
int pin[] = {2, 3, 4, 5, 6, 7, 8};
int pinOffset = 2;     //Première LED sur broche 2
int buttonPin = 13;    //Bouton-poussoir sur broche 13

void setup(){
  for(int i = 0; i < 7; i++)
    pinMode(pin[i], OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop(){
  if (digitalRead (buttonPin) == HIGH)
     displayPips(random (1, 7)); //Générer un nombre entre 1 et 6
}

void displayPips(int value){
  for(int i = 0; i < 7; i++)
    digitalWrite(i + pinOffset,(pips[value - 1][i] == 1)?HIGH:LOW);
  delay(WAITTIME);          //Ajouter une courte pause
}
```

Revue de code

Du point de vue logiciel, les variables présentées dans le tableau 8-2 sont nécessaires à notre montage.

Tableau 8-2 ▶
Variables nécessaires et leur objet

Variable	Objet
pips	Tableau bidimensionnel contenant les informations sur les LED à commander pour la valeur d'affichage respective
pin	Tableau unidimensionnel contenant les numéros des différentes broches de LED
pinOffset	La première LED ne se trouve pas sur la broche 0. Cette variable contient une valeur de décalage qui définit la position de départ pour une boucle `for`, afin de commander la première LED et toutes les autres.
buttonPin	Broche de raccordement du bouton-poussoir au dé

La programmation est déjà plus compliquée et nous n'avons pas seulement affaire cette fois à un tableau unidimensionnel comme dans le montage n° 5 sur le séquenceur de lumière. Un tableau bidimensionnel est ici nécessaire pour mémoriser les numéros des LED qui doivent s'allumer en fonction du nombre obtenu. Rappelons-nous encore une fois, par l'intermédiaire de la figure 8-2, comment un tableau unidimensionnel fonctionne et comment nous pouvons y accéder.

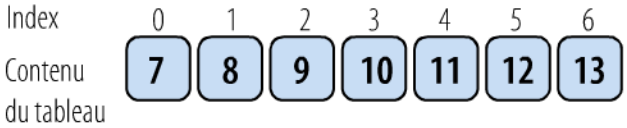


Figure 8-2
Tableau unidimensionnel

La déclaration et l'initialisation du tableau sont assurées par la ligne suivante :

```
int ledPin[] = {7, 8, 9, 10, 11, 12, 13};
```

Le tableau contient ici sept éléments. Un tableau unidimensionnel est reconnaissable à sa paire de crochets derrière le nom de variable. On accède à un élément particulier en indiquant l'index entre les crochets. Vous écrivez donc ce qui suit pour accéder au 4e élément :

```
ledPin[3]
```

N'oubliez pas que l'on compte à partir de 0 ! Un tableau bidimensionnel possède au sens figuré une dimension spatiale de plus, passant ainsi quasiment d'une droite unidimensionnelle à une surface.

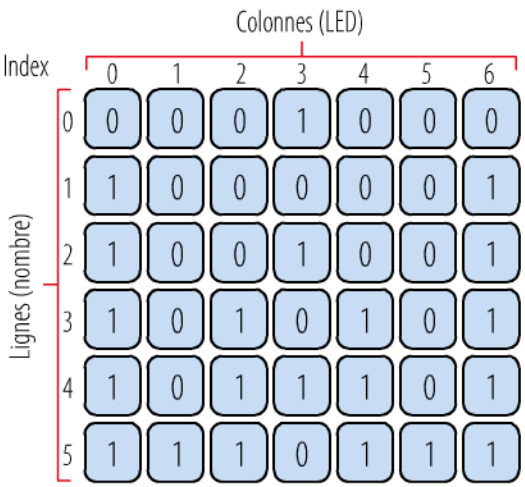


Figure 8-3
Tableau bidimensionnel

Il se comporte de la même manière que pour trouver une pièce sur un jeu d'échec. On la localise plus facilement grâce à des coordonnées : par exemple Dame sur D1, D indiquant la colonne et 1 la rangée. Le tableau présenté ici dispose de $6 \times 7 = 42$ éléments. Déclaration et initialisation se font comme d'habitude. Seule la paire de crochets est rajoutée pour la nouvelle dimension.

```
int pips[6][7] = {{0, 0, 0, 1, 0, 0, 0},     //Nombre sorti 1
                  {0, 0, 1, 0, 0, 0, 1},     //Nombre sorti 2
                  {0, 0, 1, 1, 0, 0, 1},     //Nombre sorti 3
                  {1, 0, 1, 0, 1, 0, 1},     //Nombre sorti 4
```

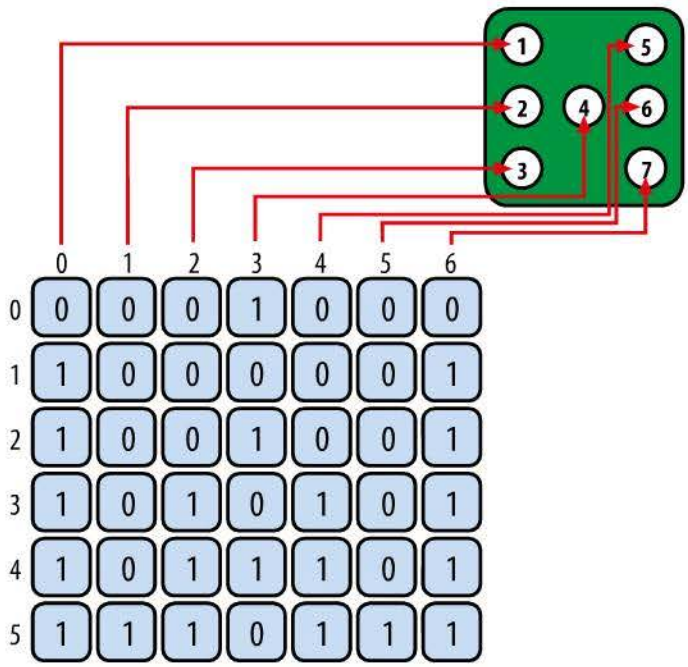
```
                {1, 0, 1, 1, 1, 0, 1},    //Nombre sorti 5
                {1, 1, 1, 0, 1, 1, 1}};   //Nombre sorti 6
```

La première valeur [6] entre crochets indique le nombre de lignes, le deuxième [7] le nombre de colonnes. La double paire de crochets permet aussi d'accéder à un élément :

```
pips[ligne][colonne]
```

Vous pouvez ainsi procéder ligne par ligne et lire les valeurs de LED correspondantes pour y accéder. La figure 8-4 montre l'affectation des différentes valeurs.

Figure 8-4 ▶
Affectation des valeurs des colonnes du tableau aux LED correspondantes



Il y a quelque chose que je trouve bizarre. On ne peut pas faire 0 avec un dé. Pourtant le graphique, lui, commence par 0 et finit par 5 à la place de 6. Pouvez-vous m'expliquer ?

La réponse est simple. Vous avez un peu tout mélangé… Ce ne sont pas les points du dé qui sont énumérés mais l'index du tableau. Rappelez-vous que l'index commence toujours à 0 et présente donc un décalage numérique de – 1 par rapport aux points du dé. Voici maintenant un petit sketch qui affiche les contenus du tableau bidimensionnel sur le Serial Monitor :

```
int pips[6][7] = {{0, 0, 0, 1, 0, 0, 0},    //Nombre sorti 1
                  {1, 0, 0, 0, 0, 0, 1},    //Nombre sorti 2
```

```
                {1, 0, 0, 1, 0, 0, 1},    //Nombre sorti 3
                {1, 0, 1, 0, 1, 0, 1},    //Nombre sorti 4
                {1, 0, 1, 1, 1, 0, 1},    //Nombre sorti 5
                {1, 1, 1, 0, 1, 1, 1}};   //Nombre sorti 6

void setup(){
  Serial.begin(9600);
  for(int row = 0; row < 6; row++){
    for(int col = 0; col < 7; col++)
      Serial.print(pips[row][col]);
    Serial.println();
     }
}
void loop(){…}
```

Il s'agit ici de deux boucles `for` imbriquées. La boucle extérieure, qui contient la variable de contrôle `row` (ligne), commence à compter à partir de sa valeur de départ `0`. Vient ensuite la boucle intérieure, qui commence elle aussi par la valeur `0` de sa variable de contrôle `col` (colonne). La boucle intérieure doit cependant avoir fini de traiter toutes ses valeurs pour que la boucle extérieure incrémente la sienne.

Pour aller plus loin

Dans le cas de boucles imbriquées l'une dans l'autre, le traitement se fait de l'intérieur vers l'extérieur. Autrement dit, la boucle intérieure doit avoir exécuté tous ses passages avant que la boucle extérieure ne compte un de plus et que la boucle intérieure ne poursuive avec ses passages. Le cycle continue jusqu'à ce que toutes les boucles aient été traitées.

La figure 8-5 présente le contenu du tableau imprimé sur le Serial Monitor.

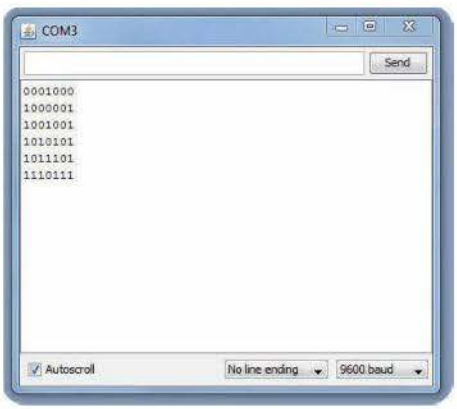


Figure 8-5
Contenu du tableau imprimé ligne par ligne sur le Serial Monitor

Comparez cette impression à l'initialisation du tableau et vous verrez qu'elles coïncident. Passons maintenant à l'analyse du code propre-

ment dite. La fonction `setup` a encore pour tâche d'initialiser les différentes broches :

```
void setup(){
  for(int i = 0; i < 7; i++)
    pinMode(pin[i], OUTPUT);
  pinMode(buttonPin, INPUT);
}
```

Les broches pour commander les LED, broches programmées comme `OUTPUT` dans la fonction `setup`, sont également regroupées dans un tableau. Il n'y a qu'au bouton-poussoir, qui est relié à une entrée numérique, qu'une variable normale est affectée. La tâche principale est encore exécutée par la fonction `loop`.

```
void loop(){
  if(digitalRead(buttonPin) == HIGH)
    displayPips(random (1, 7));  //Générer un nombre entre 1 et 6
}
void displayPips(int value){
  for(int i = 0; i < 7; i++)
    digitalWrite(i + pinOffset, (pips[value-1][i] == 1)?HIGH:LOW);
  delay(WAITTIME);
}
```

Quand le bouton-poussoir est enfoncé, la fonction `displayPips` est appelée. Un chiffre aléatoire compris entre 1 et 6 lui est transmis comme argument. Voyons maintenant de plus près le mode d'exploitation de la fonction. Il s'agit essentiellement d'une boucle `for`, qui commande les différentes LED correspondant au chiffre transmis.

Supposons qu'un 4 soit sorti : la fonction reçoit cette valeur comme argument. La boucle `for` commence son travail. Elle commande les broches et détermine le niveau `HIGH/LOW` nécessaire pour la LED en question :

```
for(int i = 0; i < 7; i++)
    digitalWrite ( + pinOffset, (pips[ value - 1][ i]  == 1)?HIGH:LOW);
```

Broche de la LED — Niveau HIGH/LOW

C'est la variable de décalage (`offset`) qui est utilisée, mais je n'ai pas très bien compris à quoi elle sert.

Pas de problème, Ardus ! La variable `pinOffset` a pour valeur `2` et établit que la première broche à traiter se trouve à cette place. La première broche, portant le numéro `0`, est `RX` et la deuxième, portant le

numéro 1, est `TX`. Ces deux broches sont en principe à éviter. La boucle `for` commençant par la valeur `0`, la valeur de `pinOffset` lui est rajoutée. Mais revenons à notre exemple, dans lequel un `4` est sorti. La boucle `for` traite la 4e ligne du tableau pour déterminer les niveaux `HIGH/LOW` nécessaires. Mais cette valeur doit être diminuée de 1 du fait que l'on commence par la valeur d'index `0`.

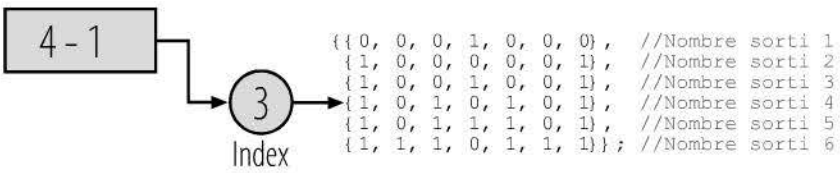


Figure 8-6
Sélection de l'élément du tableau pertinent pour un nombre préalablement sorti

La ligne sélectionnée dans le tableau comporte les valeurs 1, 0, 1, 0, 1, 0, 1 qui sont traitées une à une par la boucle `for`. Ceci est initié par l'expression suivante :

```
(pips[Value - 1][i] == 1)?HIGH:LOW)
```

Celle-ci vérifie que les valeurs sont bien `1` ou `0`. Le niveau `HIGH` est appliqué si c'est `1` et le niveau `LOW` si c'est `0`. Les LED correspondant au nombre sorti sont ainsi activées ou désactivées. Tant que le bouton-poussoir est maintenu enfoncé, un nouveau nombre est déterminé et les LED clignotent toutes très vite l'une derrière l'autre. Une fois le bouton-poussoir relâché, le dernier nombre reste affiché. La constante `WAITTIME` permet de régler la vitesse à laquelle les nombres changent quand le bouton-poussoir est enfoncé, soit ici 20 ms.

Schéma

Le schéma montre les 7 LED du dé avec leurs résistances série de 330 Ω et le bouton-poussoir avec sa résistance pull-down.

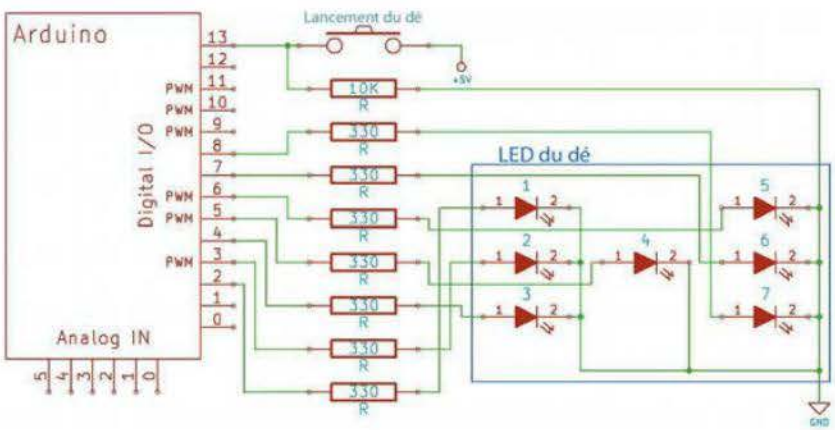
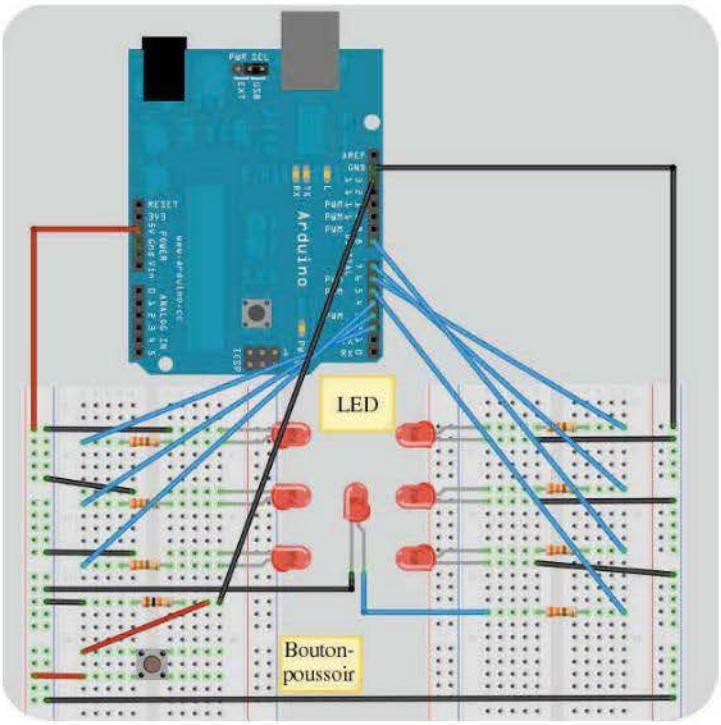


Figure 8-7
Carte Arduino commandant chacune des 7 LED de notre dé

Réalisation du circuit

Figure 8-8 ▶
Réalisation du dé électronique avec Fritzing



On voit que j'ai utilisé deux plaques d'essais pour construire ce circuit. Il existe cependant des versions suffisamment larges pour pouvoir placer tous les composants dessus. Faites des essais de disposition car vous n'êtes pas obligé de suivre ce que j'ai fait. À vous de trouver votre propre stratégie. La figure 8-9 montre la construction du circuit sur une seule plaque, mais il a fallu « jongler » un peu pour y arriver. Ceci dit, il devrait très bien fonctionner.

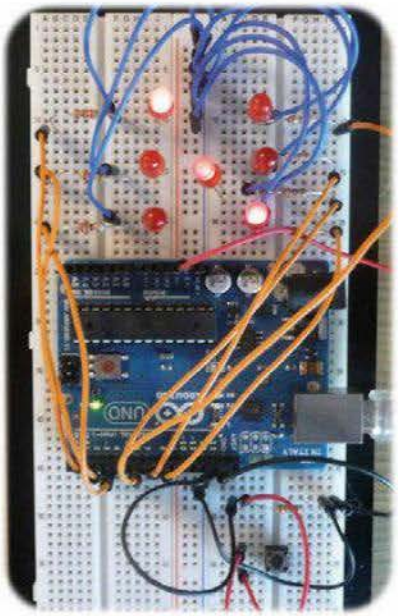


Figure 8-9
Réalisation du dé électronique sur une plaque d'essais

Il m'est venu une idée. Je me suis souvenu du tableau unidimensionnel et j'ai essayé quelque chose. Vous avez dit qu'il est inutile d'écrire la dimension du tableau entre les crochets si celui-ci est initialisé aussitôt dans la même ligne. Le compilateur déduirait alors des valeurs transmises les dimensions que doit avoir le tableau. J'ai donc voulu essayer avec le tableau bidimensionnel mais j'ai eu une erreur.

L'idée n'est pas mauvaise et prouve que vous y pensez et appliquez ce que vous avez appris. Mais les choses ne vont pas si bien avec le tableau bidimensionnel. Si vous omettez toutes les indications sur la taille du tableau et écrivez :

```
int pips[][] = {{0, 0, 0, 1, 0, 0, 0},    //Nombre sorti 1
                {1, 0, 0, 0, 0, 0, 1},    //Nombre sorti 2
                {1, 0, 0, 1, 0, 0, 1},    //Nombre sorti 3
                {1, 0, 1, 0, 1, 0, 1},    //Nombre sorti 4
                {1, 0, 1, 1, 1, 0, 1},    //Nombre sorti 5
                {1, 1, 1, 0, 1, 1, 1}};   //Nombre sorti 6
```

le compilateur renâcle, comme vous avez pu le constater.

Le message d'erreur dit, pour résumer, que dans le cas d'un tableau multidimensionnel, toutes les limites, hormis la première, doivent être indiquées. Vous pouvez donc écrire la ligne suivante :

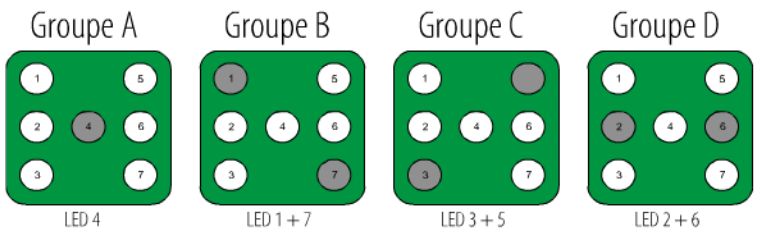
```
int pips[][7] = …
```

Le compilateur acceptera ce code.

Que pouvons-nous encore améliorer ?

Il est presque toujours possible d'améliorer ou de simplifier les choses. Il vous suffit de prendre un peu de recul et de considérer le projet dans son ensemble. Ne vous creusez pas trop la tête. Les idées viennent souvent quand on est occupé à autre chose. Revenons à notre dé. Si vous regardez les différents points d'un dé pour différentes valeurs, vous remarquerez peut-être quelque chose. Retournez pour ce faire au tableau « Quelle LED s'allume pour quel nombre ? » Question : les huit LED s'allument-elles toutes indépendamment les unes des autres ? Ou se peut-il que certaines forment un groupe ? Question idiote, non ? C'est le cas, bien évidemment : la figure 8-10 montre les différents groupes.

Figure 8-10 ▶
Groupes de LED sur le dé électronique



Pris séparément, le groupe A et le groupe B sont utilisables, ce qui est moins le cas pour les groupes C et D.

Quoi qu'il en soit, les configurations souhaitées sont générées par un groupe ou une combinaison de plusieurs groupes. Voyons maintenant lequel ou lesquels des groupes est ou sont concerné(s) par quels points du dé :

Tableau 8-3 ▶
Points du dé et groupes de LED

Dé						
Groupe A	✓		✓		✓	
Groupe B		✓	✓	✓	✓	✓
Groupe C				✓	✓	✓
Groupe D						✓

Ainsi, nous pouvons contrôler les LED avec 4 lignes de commandes au lieu de 7.

Si je comprends bien, il faut interconnecter deux LED dans les groupes B, C, et D. Ne peut-on pas faire autrement ? Dois-je les câbler en série ou en parallèle ?

C'est tout à fait ça, Ardus ! Dans le montage n° 2, nous avions calculé la résistance série pour une LED rouge. Relisez-le si besoin. Si plusieurs LED doivent être commandées, il faut les brancher en série. Il y a environ 2 V aux bornes d'une seule LED rouge, autrement dit la résistance série doit faire chuter 3 V. Deux LED étant ici branchées l'une derrière l'autre, il est possible de déterminer ce qui suit pour la chute de tension aux bornes de la résistance série R_V :

$$U_{RV} = U_{totale} - U_{LED1} - U_{LED2} = +5\text{ V} - 2\text{ V} - 2\text{ V} = 1\text{ V}$$

1 V doit donc être « grillé » sur la résistance série R_V pour que 2 V subsistent aux bornes de chaque LED. Pour ce qui est du courant, qui circule de la même façon dans tous les composants (rappelez-vous comment le courant se comporte dans un montage en série), je le fixe à 10 mA (10 mA = 0,01 A). On obtient donc les valeurs suivantes dans la formule pour calculer la résistance série :

$$R_V = \frac{U_{totale} - U_{LED1 + LED2}}{I} = \frac{5\text{ V} - 4\text{ V}}{0{,}01\text{ A}} = 100\ \Omega$$

Le circuit ressemble à ceci :

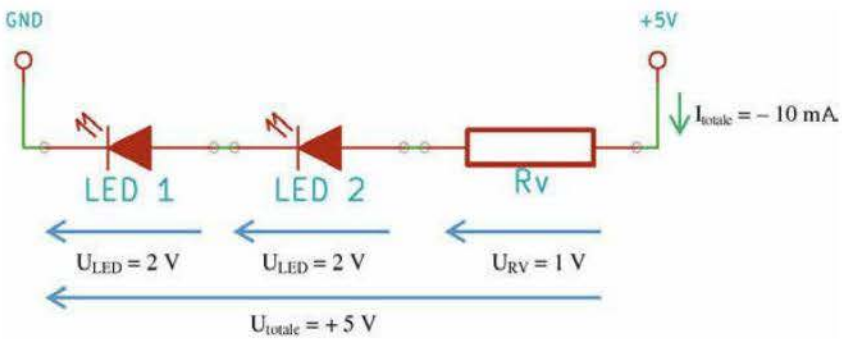
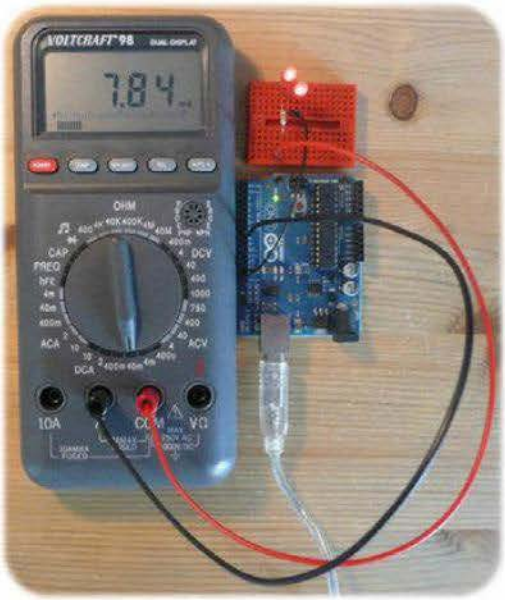


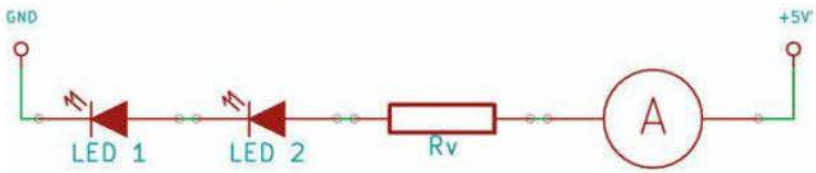
Figure 8-11
Deux LED avec une résistance série

Attention !
Veillez à ce que les deux LED soient dans le même sens, sinon pas d'allumage. L'anode de la LED 1 est reliée à la cathode de la LED 2.

Ici aussi, j'ai vérifié le calcul de manière pratique pour m'assurer que tout est également en ordre.

Figure 8-12 ▶
Mesure du courant sur le circuit de commande avec deux LED et une nouvelle résistance





Le courant de 7,84 mA est absolument correct et encore inférieur à la prescription de 10 mA maxi. Deux LED ayant naturellement besoin du double de tension d'alimentation par rapport à une seule, la résistance série doit être plus faible pour que la luminosité des deux LED, soit la même que celle d'une seule LED. Vous pouvez bien sûr utiliser la même résistance de 330 Ω pour tous les groupes A à D, ce qui signifie toutefois en théorie que la luminosité du groupe A sera plus forte avec une seule LED que le reste des groupes.

Passons maintenant à la programmation. Par quoi commencer ? Je vous suggère de revenir au tableau « Points du dé et groupes de LED » et de voir s'il s'en dégage une systématique établissant quel groupe de LED doit être commandé, à quel moment et pour quelles configurations de points du dé. Procédez étape par étape et observez un groupe après l'autre. Vous pouvez les traiter complètement à part l'un de l'autre car la logique de commande se charge ensuite de les rassembler pour un affichage en commun des véritables points du dé. Je vous montre encore une fois de manière simplifiée le groupe A du dernier tableau.

Dé	1	2	3	4	5	6
Groupe A	✓		✓		✓	

Encore un indice : qu'est-ce que les nombres 1, 3 et 5 ont en commun ?

Ce sont tous des nombres impairs.

Oui Ardus ! C'est la solution.

Formulation pour commander le groupe A

Commander le groupe A si le nombre aléatoire déterminé est impair. Passons maintenant au groupe B. Voici l'extrait correspondant du tableau :

Dé	1	2	3	4	5	6
Groupe B		✓	✓	✓	✓	✓

Que constatez-vous ici ?

Tous les nombres sont concernés sauf le 1.

Bien, Ardus ! Mais à quoi pourrait bien ressembler une formulation que le microcontrôleur comprendrait sans problème ? Une description quelque peu maladroite donnerait ceci : commander le groupe B si le nombre est 2 ou 3 ou 4 ou 5 ou 6. Cherchez là encore le point commun et vous pourrez raccourcir fortement la formulation.

Formulation pour commander le groupe B

Commander le groupe B si le nombre aléatoire déterminé est supérieur à 1. Voyons maintenant le groupe C :

Dé	1	2	3	4	5	6
Groupe C				✓	✓	✓

Vous savez comment faire maintenant, n'est-ce pas ?

Tous les nombres supérieurs à 3 sont concernés.

Oui Ardus !

Formulation pour commander le groupe C

Commander le groupe C si le nombre aléatoire déterminé est supérieur à 3. Passons pour finir au groupe D :

Dé	1	2	3	4	5	6
Groupe D						✓

Plus besoin de vous demander maintenant, n'est-ce pas ?

Formulation pour commander le groupe D

Commander le groupe D si le nombre aléatoire déterminé est égal à 6. Nous pouvons à présent passer à la programmation proprement dite. Vous verrez que cette solution est beaucoup plus simple que l'utilisation d'un tableau. Mais il faut suivre mentalement quelques pistes jusqu'au bout avant de s'apercevoir que 4 broches au lieu de 7 sont nécessaires pour commander les LED. Cela permet cependant d'aborder cette thématique par le côté ludique. Voici le code du sketch pour commander le dé électronique avec moins de lignes de commande :

```
#define WAITTIME 20
int GroupA = 8;    //LED 4
int GroupB = 9;    //LED 1 + 7
int GroupC = 10;   //LED 3 + 5
int GroupD = 11;   //LED 2 + 6
int buttonPin = 13; //Bouton-poussoir à la broche 13
void setup(){
  pinMode(GroupA, OUTPUT);
  pinMode(GroupB, OUTPUT);
  pinMode(GroupC, OUTPUT);
  pinMode(GroupD, OUTPUT);
}

void loop(){
  if(digitalRead(buttonPin) == HIGH)
```

```
    displayPips(random(1, 7)); //Générer un nombre entre 1 et 6
}

void displayPips(int value){
  //Éteindre tous les groupes
  digitalWrite(GroupA, LOW);
  digitalWrite(GroupB, LOW);
  digitalWrite(GroupC, LOW);
  digitalWrite(GroupD, LOW);
  //Commande de tous les groupes
  if(value%2 != 0)    //La valeur est-elle impaire ?
   digitalWrite(GroupA, HIGH);
  if(value > 1)
   digitalWrite(GroupB, HIGH);
  if(value > 3)
   digitalWrite(GroupC, HIGH);
  if(value == 6)
   digitalWrite(GroupD, HIGH);
delay(WAITTIME);    //Ajouter une courte pause
}
```

Je viens de m'apercevoir que vous avez oublié quelque chose ! Vous avez programmé les broches pour les groupes A à D comme sortie, mais vous avez oublié de définir le bouton-poussoir comme entrée.

C'est vrai, Ardus ! Je n'ai pas programmé cette entrée sur la broche 13 comme entrée. Vous avez raison sur ce point. Mais je n'ai pas non plus oublié puisque toutes les broches numériques sont définies comme entrée de manière standard et n'ont donc pas besoin d'être explicitement programmées en cas d'utilisation appropriée.

Vous pouvez bien entendu le faire partout dans votre sketch car cela aide certainement à comprendre.

J'ai en fait tout compris jusqu'à la ligne dans laquelle il est déterminé si la valeur est impaire. Pouvez-vous m'expliquer s'il vous plaît ?

Bien sûr, Ardus ! L'opérateur `%` (opérateur modulo) détermine toujours le reste d'une division. Si le nombre est divisible par 2, il s'agit d'un nombre pair. Le reste de la division étant dans ce cas toujours `0`. La ligne :

```
if(value%2 != 0)
```

me permet cependant de demander si le reste est différent de `0` pour ainsi commander le groupe A.

Encore une remarque avant d'en venir au circuit : cela ne change pas grand-chose si, au lieu de la résistance série de 100 Ω calculée pour les groupes B à D, vous utilisez ici les anciennes résistances de 330 Ω. La luminosité semble être la même, mais ce n'est qu'approximatif. La figure 8-13 montre qu'il faut moins de résistances série pour les LED que dans le montage précédent.

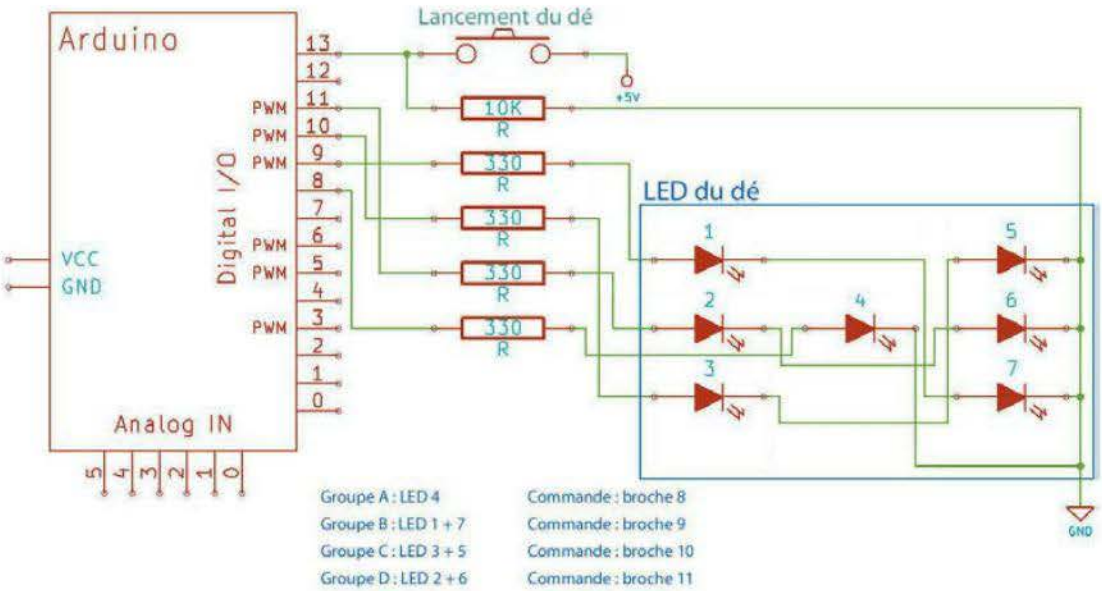


Figure 8-13 ▲
Carte Arduino commandant les 7 LED de notre dé par groupe de LED

La réalisation sur la plaque d'essais s'avère plus simple parce que les lignes de commande sont moins nombreuses :

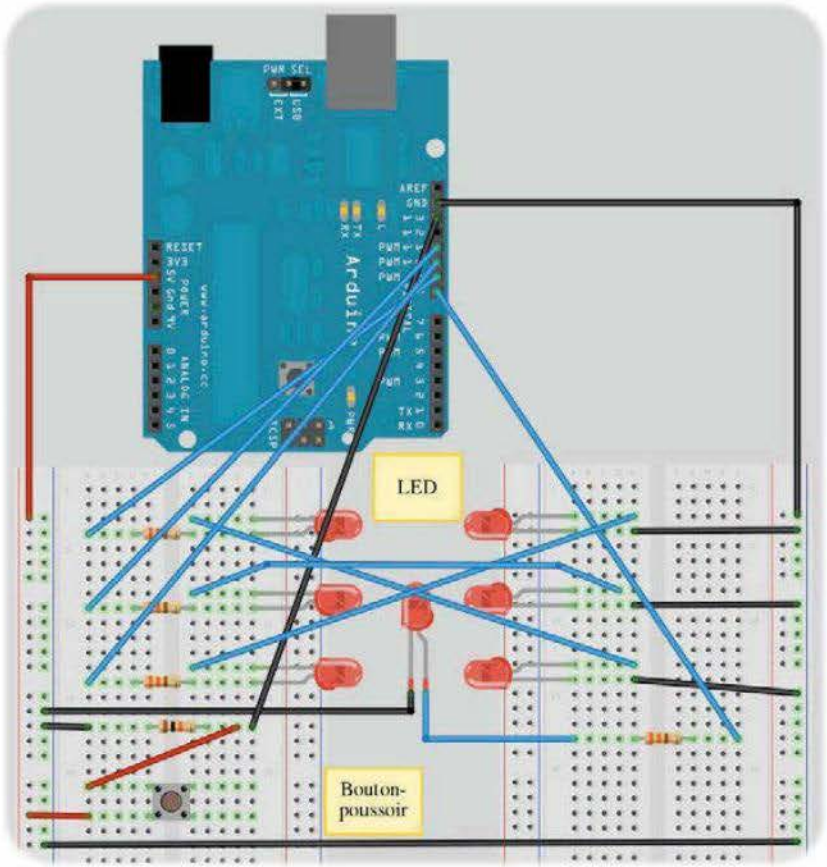


Figure 8-14
Réalisation du dé électronique utilisant des groupes de LED avec Fritzing

Problèmes courants

Si les LED ne se mettent pas à clignoter après avoir appuyé sur le bouton-poussoir ou si les points du dé qui s'affichent sont bizarres, voire incohérents, débranchez le port USB de la carte pour plus de sécurité et vérifiez ce qui suit.

- Vos fiches de raccordement sur la maquette correspondent-elles vraiment au circuit ?
- N'y a-t-il pas un court-circuit éventuel entre elles ?
- Les LED ont-elles été mises dans le bon sens ? Autrement dit, la polarité est-elle correcte ?
- Les résistances ont-elles bien les bonnes valeurs ?
- Le code du sketch est-il correct ?
- Le bouton-poussoir est-il correctement câblé ? Vérifiez encore une fois les contacts en question avec un testeur de continuité.

Qu'avez-vous appris ?

- Vous avez appris dans ce montage comment déclarer et initialiser un tableau bidimensionnel et comment accéder aux différents éléments de ce tableau.
- Vous savez comment imprimer des contenus de variables avec le Serial Monitor pour vérifier l'exactitude de ces valeurs. Vous pouvez ainsi rechercher une erreur et analyser le code en cas de comportement incorrect. Vous devez cependant être sûr que le circuit est correctement câblé, sinon vous chercherez dans le code-source une erreur qui, en réalité, se situe au niveau du matériel. Cela vous évitera de perdre du temps et vous épargnera peut-être même une crise de nerf.
- Vous avez appris comment calculer une résistance série pour deux LED montées en série, de manière à ce que la luminosité demeure pratiquement inchangée.

Exercice complémentaire

L'objet de cet exercice est déjà un peu plus délicat. Vous vous souvenez sûrement du registre à décalage 74HC595 avec ses 8 sorties. Essayez de réaliser un circuit ou de programmer un sketch qui commande un dé électronique au moyen du registre à décalage. Combien de broches numériques économisez-vous avec cette variante ? Est-ce un avantage par rapport à la réalisation avec des groupes de LED ?