Montage

16 Le son et plus encore

Au sommaire :

- l'émission de son au moyen d'un élément piézoélectrique ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- la création du jeu de la séquence des couleurs ;
- un exercice complémentaire.

Y a pas le son ?

À la longue, vous en avez peut-être assez des signaux lumineux et autres LED clignotantes. Aussi allons-nous voir à présent comment votre carte Arduino peut émettre des sons au moyen d'un élément piézoélectrique. Ce composant vous a déjà été présenté dans le chapitre 4 sur les bases de l'électronique.

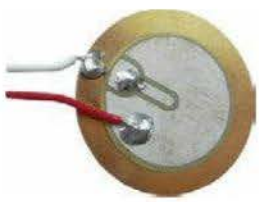


Figure 16-1
Disque piézo

Vous ne risquez pas les ondes de choc acoustiques avec un piézo, car les vibrations émises couvrent un espace des plus réduits. Il est cependant parfait pour ce que nous allons faire.

Si on branche, par exemple, l'élément sur une sortie numérique et si on passe à intervalles réguliers la sortie en niveau `HIGH` ou `LOW`, on entend un craquement dans l'élément piézoélectrique. Plus le laps de

temps entre niveaux `HIGH` et `LOW` est court, plus le son audible est aigu ; plus le laps de temps est long, plus le son est grave. Le phénomène est le même quand, par exemple, vous passez vos doigts plus ou moins vite sur une grille à lamelles. Plus vous allez vite, plus le bruit est aigu ; le piézo fonctionne sur ce principe. Un craquement répété, tantôt plus lent, tantôt plus rapide, influe sur la fréquence du son. Voici un sketch très simple pour émettre un son.

```
#define piezoPin 13  //Élément piézoélectrique sur broche 13
#define DELAY 1000

void setup(){
  pinMode(piezoPin, OUTPUT);
}

void loop(){
  digitalWrite(piezoPin, HIGH); delayMicroseconds(DELAY);
  digitalWrite(piezoPin, LOW); delayMicroseconds(DELAY);
}
```

Ne vous inquiétez pas pour la fonction `delayMicroseconds`. Son action est la même que celle de la fonction `delay`, à ceci près que la valeur transmise n'est pas interprétée en millisecondes mais en microsecondes ; la microseconde est 1 000 fois plus petite (1 ms = 1 000 µs). Cette nouvelle fonction est utilisée, car `delay` ne permet de descendre en dessous d'1 ms.

Composants nécessaires

	1 élément piézoélectrique
	Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

Pour le premier sketch utile qui doit être capable d'émettre plusieurs sons à des fréquences différentes, mieux vaut créer un tableau des sons avec différentes valeurs que nous appellerons l'une après l'autre pendant le sketch. On utilise pour ce faire la fonction `tone` (sons) mise à disposition par Arduino. Vous en saurez bientôt plus.

```
#define piezoPin 13          //Élément piézoélectrique sur broche 13
#define toneDuration 500     //Durée du son
#define tonePause 800        //Longueur de la pause entre les sons
int tones[] = {523, 659, 587, 698, 659, 784, 698, 880};
int elements = sizeof(tones) / sizeof(tones[0]);

void setup(){
  noTone(piezoPin);   //Rendre le piézo muet
  for(int i = 0; i < elements; i++){
    tone(piezoPin, tones[i], toneDuration); //Exécuter le son
    delay(tonePause); //Pause entre les sons
  }
}

void loop(){/* vide */}
```

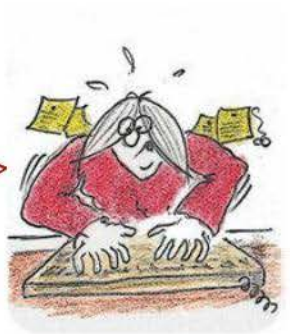
Revue de code

Du point de vue logiciel, les variables suivantes sont nécessaires à notre expérimentation.

Tableau 16-1
Variables nécessaires et leur objet

Variable	Objet
`tones[]`	Tableau contenant les fréquences des différents sons à exécuter
`elements`	Nombre d'éléments du tableau

Le tableau unidimensionnel `tones` est du type de donnée `int` et contient les fréquences en hertz des sons à exécuter. Les hertz (Hz) servent à mesurer le nombre de vibrations par seconde. Plus la valeur est élevée, plus le son est aigu, et vice versa. Le nombre d'éléments du tableau est affecté à la variable `elements` ; il servira plus tard dans la boucle `for` pour traiter tous les éléments. Le réglage manuel de la limite supérieure ou de la condition de la boucle `for` est ainsi évité, celui-ci étant fait automatiquement au moyen d'un calcul.



Oh la la ! J'ai du mal avec le calcul des éléments du tableau. Pouvez-vous m'expliquer s'il vous plaît ?

J'allais y venir. On utilise pour ce faire la fonction `sizeof` de C++, qui détermine la taille d'une variable ou d'un objet dans la mémoire. Voici un court exemple :

```
byte byteValue = 16;    //Variable du type byte
int intValue = 4;       //Variable du type int
long longValue = 3.14;  //Variable du type long
int myArray[] = {25, 46, 9};  //Tableau du type int
```

```
void setup(){
   Serial.begin(9600);
   Serial.print("Nombre d'octets pour 'byte' : ");
   Serial.println(sizeof(byteValue));
   Serial.print("Nombre d'octets pour 'int' : ");
   Serial.println(sizeof (intValue));
   Serial.print("Nombre d'octets pour 'long' : ");
   Serial.println(sizeof(longValue));
   Serial.print("Nombre d'octets pour 'myArray' : ");
   Serial.println(sizeof(myArray));
}

void loop(){/ * vide */}
```

L'affichage est donc le suivant :

```
Nombre d'octets pour 'byte' : 1
Nombre d'octets pour 'int' : 2
Nombre d'octets pour 'long' : 4
Nombre d'octets pour 'myArray' : 6
```

Quand on regarde les valeurs pour les types de données `byte`, `int` et `long`, on s'aperçoit qu'elles sont identiques à celles indiquées dans le chapitre 9, dans lequel il était question de types de données et de domaines de valeurs.

Passons à la dernière ligne de l'affichage. On y voit que le tableau occupe 6 octets de mémoire, ce qui est logique puisqu'un seul élément `int` nécessite 2 octets de mémoire. Or, nous avons 3 éléments. Le résultat est donc $2 \times 3 = 6$ octets. La ligne :

```
int elements = sizeof(tones) / sizeof(tones[0]);
```

divise le nombre d'octets du tableau par le nombre d'octets d'un seul élément. C'est toujours de cette manière qu'on obtient le nombre d'éléments d'un tableau. Mais revenons à notre sketch. Tout au début, la fonction `noTone` rend le piézo muet au cas où il devrait encore pépier du fait d'un sketch précédent. Elle n'a qu'un seul paramètre, qui indique la broche sur laquelle se trouve le piézo.

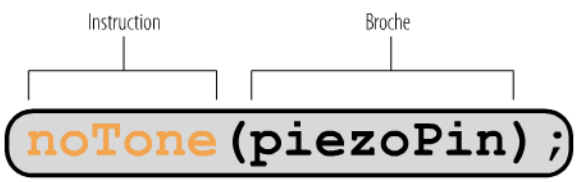


Figure 16-2 La fonction noTone rend le piézo muet.

La fonction `tone` possède en revanche deux autres paramètres. L'un indique la fréquence et l'autre la durée pendant laquelle le son doit être audible.

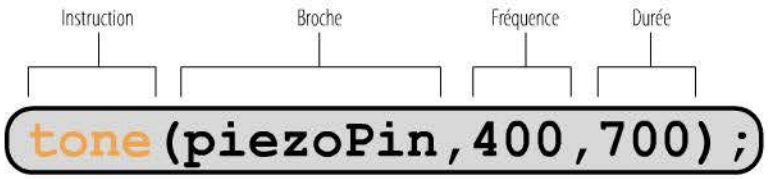


Figure 16-3
La fonction `tone` rend le piézo bavard.

Pouvez-vous me dire comment vous en êtes venu aux différentes valeurs que vous utilisez dans le tableau des sons ? Vous les avez toutes essayées pour savoir lesquelles conviennent à peu près ?

Non, elles sont tirées d'un exemple de sketch qui se trouve dans l'IDE Arduino. Recherchez le fichier `pitches.h` dans le dossier `examples` de l'installation Arduino et ouvrez-le avec un éditeur. Vous y trouverez les fréquences correspondant à de nombreuses notes. Vous pouvez inclure ce fichier dans votre sketch et utiliser ensuite directement les constantes symboliques. Essayez ! Le code est alors beaucoup plus parlant et plus clair que lorsque des valeurs numériques sont utilisées.

Réalisation du circuit

Le circuit n'a vraiment rien d'extraordinaire, pas vrai ?

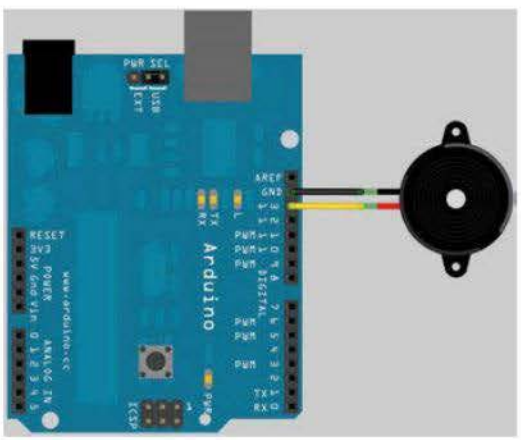


Figure 16-4
L'élément piézoélectrique connecté

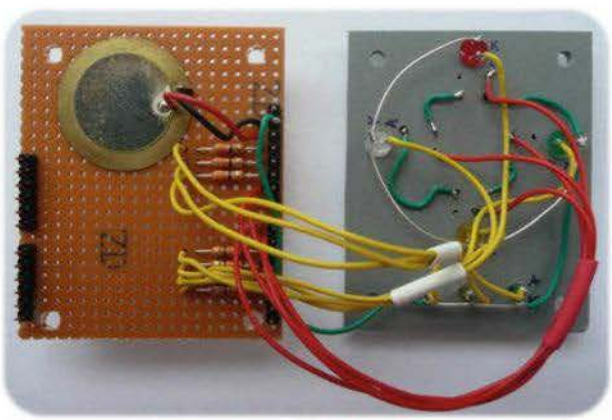
Sketch élargi : jeu de la séquence des couleurs

Venons-en maintenant à un jeu intéressant dit de la séquence des couleurs. Nous avons quatre LED de couleurs différentes disposées en carré. Près de chacune se trouve un bouton-poussoir. Le microcontrôleur imagine un motif pour l'ordre d'allumage des LED ; à vous de le reproduire correctement. Au début, la séquence ne comporte qu'une seule LED allumée ; une nouvelle vient s'ajouter après chaque bonne réponse. L'allumage de chacune des quatre LED est accompagné d'un son qui lui est propre. Le jeu ravit donc non seulement les yeux, mais aussi les oreilles. J'ai ajouté au circuit un shield avec une face avant de ma fabrication. Voyez plutôt (voir figure 16-5).

Figure 16-5 ▶
Shield avec la face avant pour le jeu de la séquence des couleurs

Sur la face avant, on voit les quatre grosses LED de 5 mm avec leur bouton-poussoir respectif. Quand une LED s'allume, on doit appuyer sur le bouton-poussoir placé à côté. La partie basse de la face avant est occupée par trois plus petites LED de 3 mm. Elles servent à afficher l'état, sur lequel je reviendrai plus tard. La figure 16-6 montre bien le shield et la face avant avec le câblage.

Figure 16-6 ▶
Shield ouvert et la face avant retournée



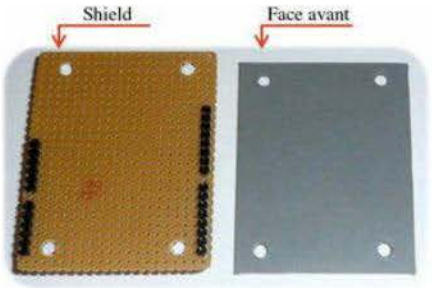
Les choses sont moins graves qu'elles n'y paraissent, et la construction devient claire quand on regarde le schéma. Voici les points que je poserai comme conditions pour le jeu :

- une longueur déterminée de la séquence, d'abord constante, est fixée par le sketch ;
- chacune des quatre LED doit avoir sa propre note avec une fréquence particulière ;
- quand une des quatre LED s'allume, la note correspondante est émise ;
- quand le bouton-poussoir situé à côté est appuyé, la LED s'allume et la note correspondante est émise ;
- si la séquence a été correctement reproduite, la LED d'état verte s'allume et une suite de sons *crescendo* se fait entendre. Le jeu reprend ensuite au début avec une nouvelle séquence ;
- si la séquence reproduite est fausse à un endroit quelconque, la LED d'état rouge s'allume et une suite de sons *decrescendo* se fait entendre. Le jeu redémarre ensuite avec une nouvelle séquence.

Composants nécessaires

	4 LED (si possible de couleurs différentes)
	7 résistances de 330 Ω
	3 LED 3 mm (rouge, verte et jaune)
	4 boutons-poussoir
	4 entretoises DK 15 mm, en plastique

4 vis M3 30 mm coupées à 23 mm environ + 4 écrous



1 shield + 1 face avant

Fils de différentes couleurs

2 barrettes à 6 broches + 2 barrettes à 8 broches

Voyons maintenant le schéma.

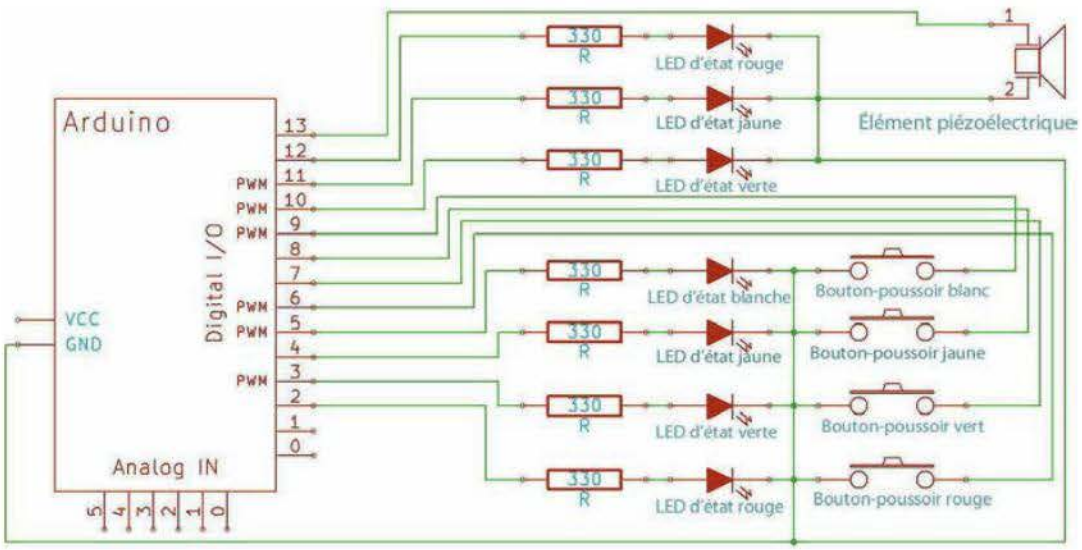


Figure 16-7▲ Circuit complet du jeu de la séquence des couleurs

Et voici maintenant le code du sketch quelque peu élargi.

```
#define MAXARRAY 5              //Définir la longueur de la séquence
int ledPin[] = {2, 3, 4, 5};    //Tableau de LED avec numéros de broche
#define piezoPin            13  //Broche piézo
#define buttonPinRed         6  //Broche bouton-poussoir LED rouge
#define buttonPinGreen       7  //Broche bouton-poussoir LED verte
#define buttonPinYellow      8  //Broche bouton-poussoir LED jaune
#define buttonPinWhite       9  //Broche bouton-poussoir LED blanche
#define ledStatePinGreen    10  //LED d'état verte
#define ledStatePinYellow   11  //LED d'état jaune
#define ledStatePinRed      12  //LED d'état rouge
int colorArray[MAXARRAY];       //Contient la suite de chiffres
                                //pour les couleurs à afficher
int tones[] = {1047, 1175, 1319, 1397};  //Fréquences des sons
                                         //pour les 4 couleurs
int counter = 0;                //Nombres de LED actuellement
                                //allumées
boolean fail = false;

void setup(){
  Serial.begin(9600);
  for(int i = 0; i < 4; i++)
    pinMode(ledPin[i], OUTPUT);  //Programmation des broches
                                 //de LED comme sortie
  pinMode(buttonPinRed,INPUT);digitalWrite(buttonPinRed,HIGH);
  pinMode(buttonPinGreen,INPUT);digitalWrite(buttonPinGreen,HIGH);
  pinMode(buttonPinYellow,INPUT);digitalWrite(buttonPinYellow,HIGH);
  pinMode(buttonPinWhite,INPUT);digitalWrite(buttonPinWhite,HIGH);
  pinMode(ledStatePinGreen, OUTPUT);
  pinMode(ledStatePinYellow, OUTPUT);
  pinMode(ledStatePinRed, OUTPUT);
}

void loop(){
  Serial.println("Départ du jeu");
  generateColors();
  int buttonCode;
  for(int i = 0; i <= counter; i++){  //Boucle extérieure
    giveSignalSequence(i);
    for(int k = 0; k <= i; k++){      //Boucle intérieure
      while(digitalRead(buttonPinRed) && digitalRead(buttonPinGreen)
&& digitalRead(buttonPinYellow) && digitalRead(buttonPinWhite));
      Serial.println ("Bouton poussé !"); //Pour contrôle dans
                                          //Serial Monitor
        //Affichage de la couleur appuyée
        if(!digitalRead(buttonPinRed))
          buttonCode = 0;
        if(!digitalRead(buttonPinGreen))
          buttonCode = 1;
        if(!digitalRead(buttonPinYellow))
          buttonCode = 2;
```

```
        if(!digitalRead(buttonPinWhite))
          buttonCode = 3;
        giveSignal(buttonCode);
        //Vérifier si la bonne couleur a été appuyée
        if(colorArray[k] != buttonCode){
          fail = true;
          break;  //Quitter la boucle for interne
        }
      }
        if(!fail)
          Serial.println("correct");  //Pour contrôle dans Serial
                                      //Monitor
        else{
          digitalWrite(ledStatePinRed, HIGH);
          for(int i = 3000; i > 500; i-=150){
           tone(piezoPin, i, 10); delay(20);
          }
          Serial.println("faux"); //Pour contrôle dans Serial
                                  //Monitor
          delay(2000);
          digitalWrite(ledStatePinRed, LOW);
          counter = 0; fail = false;
          break;                  //Quitter la boucle for
        }
          delay(2000);

        if(counter + 1 == MAXARRAY){
          digitalWrite(ledStatePinGreen, HIGH);
          for(int i = 500; i < 3000; i+=150){
            tone(piezoPin, i, 10); delay(20);
          }
          Serial.println("Fini !");    //Pour contrôle dans Serial
                                       //Monitor
          delay(2000);
          digitalWrite(ledStatePinGreen, LOW);
          counter = 0; fail = false;
          break;                  //Quitter la boucle for extérieure
        }
        counter++;                //Incrémenter le compteur
      }
}

void giveSignalSequence(int value){
  //Affichage LEDs
  for(int i = 0; i <= value; i++){
    digitalWrite(2 + colorArray[i], HIGH);
    generateTone(colorArray[i]); delay(1000);
    digitalWrite(2 + colorArray[i], LOW); delay(1000);
```

```
    }
}

void generateTone(int value){
    tone(piezoPin, tones[value], 1000);
}

void giveSignal(int value){
    //Affichage LED + Tonsignal
    digitalWrite(2 + value, HIGH); generateTone(value); delay(200);
    digitalWrite(2 + value, LOW); delay(200);
}

void generateColors(){
    randomSeed(analogRead(0));
    for(int i = 0; i < MAXARRAY; i++)
      colorArray[i] = random(4);    //Générer des chiffres aléatoires
                                    //de 0 à 3
      //0 = rouge, 1 = vert, 2 = jaune, 3 = blanc
    for(int i = 0; i < MAXARRAY; i++)
      Serial.println(colorArray[i]);    //Pour contrôle dans Serial
                                        //Monitor
}
```

Comment la programmation fonctionne-t-elle en détail ? Le sketch semble fastidieux au premier abord. Ne le considérez pas dans son intégralité, mais prenez le temps de décomposer le programme en sous-ensembles et de procéder étape par étape. Une valeur numérique est affectée à chaque couleur à afficher : `0` pour rouge, `1` pour vert, `2` pour jaune et `3` pour blanc. Un tableau peut ainsi être initialisé avec des valeurs allant de 0 à 3 ; il pourra ensuite servir à afficher les LED.

Supposons que vous ayez un tableau avec les valeurs `0`, `2`, `2`, `1` et `3`, les diodes s'allument donc dans l'ordre suivant : rouge, jaune, jaune, vert et blanc. Dans notre sketch, son nom est `colorArray` et il reçoit ses valeurs via la fonction `generateColors`. Pour les rendre visibles, la fonction `giveSignal` convertit les valeurs en signaux pour commander les LED.

```
void giveSignalSequence(int value){
    //Affichage LEDs
    for(int i = 0; i <= value; i++){
      digitalWrite(2 + colorArray[i], HIGH);
      generateTone(colorArray[i]; delay(1000);
      digitalWrite(2 + colorArray[i], LOW); delay(1000);
    }
}
```

Si la fonction doit toujours afficher la séquence des couleurs, pourquoi avons-nous encore besoin d'une variable ? Et que signifie le 2 qui est utilisé dans la fonction `digitalWrite` ? C'était quoi déjà le truc avec les magic numbers ?

Eh bien Ardus, la séquence complète ne doit pas s'afficher au début mais seulement au fur et à mesure avec, chaque fois, une couleur en plus. Le tableau des couleurs `colorArray` contient bien la séquence complète, mais la variable transmise dans `value` indique à la fonction combien d'éléments du tableau doivent être interrogés et affichés. Les quatre grandes LED étant cependant raccordées aux sorties numériques des broches 2 à 5, le chiffre 2 est quasiment un décalage qui indique la broche de démarrage quand les valeurs `0` à `3` sont ajoutées au tableau des couleurs. Vous avez bien entendu raison quand vous dites qu'il ne faut pas utiliser de magic numbers. Vous pouvez naturellement employer une constante symbolique, par exemple avec le nom `FARBPINOFFSET`.

Avant de passer à l'explication de la logique dans la fonction `loop`, je souhaiterais qu'on revienne sur la fonction `setup`. Il y a, par exemple, des broches de bouton-poussoir qui sont, bien sûr, programmées comme entrées. Pourtant, quelque chose est envoyée à ces mêmes entrées par la fonction `digitalWrite`. Pourquoi cela ?

J'utilise la possibilité d'activer les résistances pull-up présentes et connectées en interne dans le microcontrôleur. Plus besoin ainsi de connecter des résistances pull-up ou pull-down externes. J'ai déjà expliqué cela dans le montage n° 2. Si vous avez oublié, relisez-le !

Oui, je vais le faire. Quand je vois la fonction `loop`, je me dis qu'il s'en passe de belles ! Mais ce que je ne comprends pas trop, c'est le fait que la fonction `loop` s'exécute continuellement. La première boucle `for`, que vous avez qualifiée de boucle extérieure, devrait elle aussi s'exécuter continuellement. C'est pourtant elle qui – d'après ce que comprends – est chargée d'afficher la séquence en fonction de la variable `counter`.

Oui Ardus, bien vu ! La fonction `loop`, qui est une boucle sans fin, devrait normalement s'exécuter en permanence. Seulement, j'ai incorporé un arrêt qui la bloque tant qu'un des quatre boutons-poussoirs n'est pas appuyé. Voici la partie de code en question :

```
while(digitalRead(buttonPinRed) && digitalRead(buttonPinGreen) &&
      digitalRead(buttonPinYellow) && digitalRead(buttonPinWhite));
```

Les entrées numériques, auxquelles les boutons-poussoirs sont raccordés, étant reliées au +5 V à travers les résistances pull-up internes, mon interrogation doit porter sur le niveau `LOW`. Tant que toutes les entrées sont sur niveau `HIGH`, la boucle `while` exécute l'instruction qui vient aussitôt après.

C'est là tout mon problème ! Quelle instruction est exécutée au juste ? D'après le code, la ligne suivante :

```
Serial.println("Bouton poussé !");
```

devrait être exécutée. Mais ça n'a pas beaucoup de sens !

Vous avez raison, ça n'a pas beaucoup de sens ! Vous avez cependant oublié une petite chose. L'instruction, qui vient immédiatement après la boucle `while`, est le point-virgule situé tout à la fin. Il s'agit quasiment d'une instruction vide, qui fait en sorte que la boucle `while`, quand aucun des boutons-poussoirs n'est appuyé, devienne elle-même une boucle sans fin. C'est une manière élégante d'interrompre ici le déroulement du programme. Ce n'est que quand l'un ou l'autre des boutons est appuyé que la condition dans la boucle `while` n'est plus remplie et que le programme reprend son cours. Le bouton appuyé est alors identifié afin de comparer la valeur de la couleur concernée à l'élément du tableau qui vient d'être sélectionné dans la boucle intérieure. Si une concordance a été trouvée, on passe à la valeur de couleur suivante dans la séquence. En revanche, si une erreur a été commise, la variable `fail` reçoit la valeur `true`, et l'instruction `break` fait sortir prématurément de la boucle intérieure. Autrement dit, l'instruction `if` :

```
if(!fail)…
```

reprend le cours du programme en conséquence. La variable `counter` est augmentée de la valeur `1`, dans la mesure où aucune erreur n'a été commise et où la fin de la séquence n'est pas encore atteinte, si bien que la prochaine séquence affichée sera plus longue. J'ai laissé les impressions sur le Serial Monitor dans le code pour une meilleure compréhension des procédés. Elles vous donnent au début la séquence qui a été sélectionnée pour que vous puissiez faire, le cas échéant, un peu d'expérimentation. Toute explication supplémentaire est ici superflue. Lisez une fois le code de bout en bout et essayez de le comprendre.