http://www.didel.com/ info@didel.com

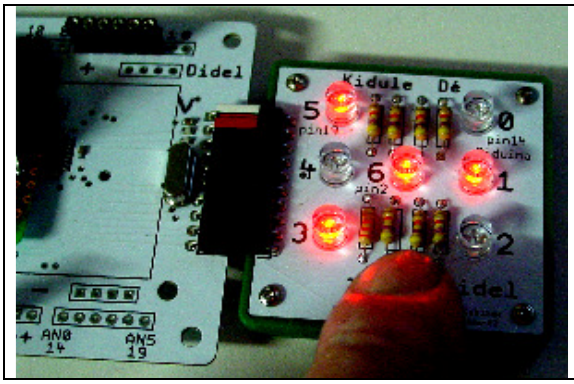


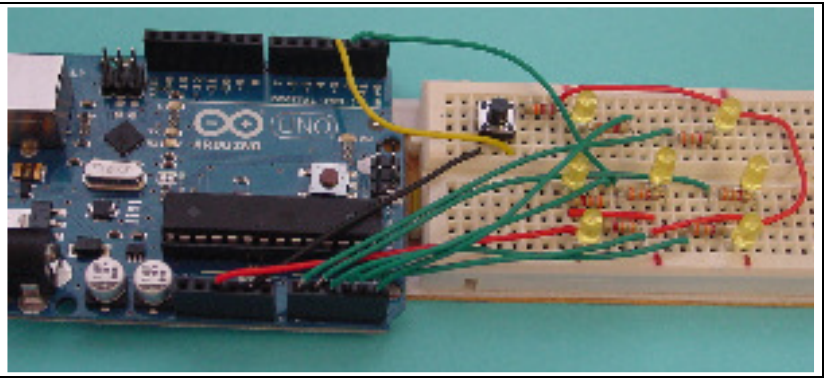
# Programmer le Kidule dé "à la Arduino"

*Note: Le "digitalWrite" d'Arduino facilite la compréhension pour les débutants, mais on devrait définir des noms explicites Led1On Led1Off qui allègeraient la lecture. Le fichier de définitions serait caché dans un fichier DeDef.h et les programmes seraient plus lisibles, tenant en général sur un seul écran ou page.*

**Matériel nécessaire**

Vous devez avoir le kidule dé ou un montage équivalent avec un poussoir (actif à 0) et 7 LEDs (actives à 0). Les programmes sont compatibles avec le Diduino et les Arduinos sompatibles.
Les programmes se trouvent sous www.didel.com/kidules/CKiDe.zip
Ce document suppose que les notions de base de la programmation C/Arduino sont acquises.
Les liens pour se former et progresser se trouvent dans www.didel.com/kidules/Liens.pdf



## Le processeur

Arduino cache l'intérieur du processeur et ne montre que ses broches (pins) qui sont liées aux ports, parfois incomplets ou avec des pins réservées. Ce qui nous intéresse c'est de travailler avec des mots de 8 bits que l'on "usine" en mémoire, et que l'on transfère dans un port 8 bits. Comprendre un peu cette structure interne d'un processeur est essentiel si on veut maîtriser les applications.
Arduino n'ayant pas de port 8 bits complet utilisable, un port compatible Microdule, le PortM (aussi appelé PortK), a été ajouté. La configuration de ce PortM suppose des connaissances non essentielles si le but est d'apprendre à programmer. Arduino cache ces correspondances entre pins et bits des registres. Accéder directement aux bits des registres est plus efficace et compatible avec tous les compilateurs C.

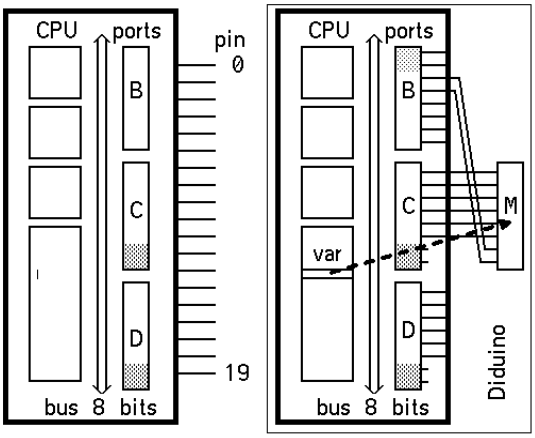
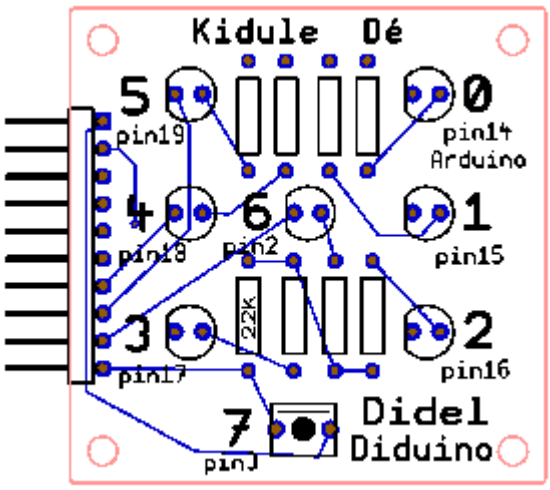


**Schéma du dé**

Le schéma est très simple. Les LEDs sont reliées au +5V via des résistances. Elle sont allumées s'il y a un 0 sur le connecteur. Le poussoir est aussi relié au +5V via une résistance. Presser sur le poussoir met la pin correspondante à zéro.
Le connecteur a comme premières pins le Gnd (0V) et le +5V. Les pins suivantes correspondent aux bits 0 .. 7 du PortK. Avec le câblage de la carte Diduino ou de l'adaptateur ArduiKi ces pins correspondent aux pins Arduino 14 à 19 puis 2 et 3. Le câblage des LEDs intervient alors.
Comme on le voit sur le dessin, on tourne dans le sens des aiguilles d'une montre et on finit au centre. Le poussoir est sur la dernière pin.




**Approche Arduino**

Pour que le programme fasse ce que l'on veut, par exemple allumer la LED du centre, il faut dire au processeur quel est le numéro de cette pin, dire que c'est une sortie et dire qu'il faut la mettre à 0 (LOW) pour allumer. On retrouve les 3 parties de tout programme: définitions, configuration, exécution. Ce qui avec Arduino s'écrit :

```
#define Led
Centre 2
```

```
void setup () {
   pinMode (LedCentre,OUTPUT);
}
```

```
void loop () {
   digitalWrite (LedCentre, LOW);
}
```

Si on numérote les LEDs comme on l'a fait de 0 à 6, on peut se référer à ces numéros, ce qui suppose qu'on a la figure précédente sous les yeux pour ne pas se tromper de numéro.

```
#define Led6 2
```

```
void setup () {
   pinMode (Led6,OUTPUT);
}
```

```
void loop () {
   digitalWrite (Led6, LOW);
}
```

Les programmes Arduino définissent souvent les noms des pins par `int LedCentre = 2 ;` ce qui revient à bloquer une variable 16 bits. Le `#define` n'est pas une instruction (pas de ; à la fin), il ne prend pas de place en mémoire.
Les programmes Arduino utilisent souvent directement les numéros de pins dans le programme, sans leur avoir donné un nom. On se souvient de ce que fait cette pin dans un programme de quelques instructions, pas dans 20 lignes où il y a d'autres numéros de pins. Ce n'est acceptable que pour de petits tests que l'on ne veut pas conserver.
De même, le LOW dans le programme pour dire que la LED est allumée dépend du câblage du dé. Une définition initiale permet dans le programme d'écrire `digitalWrite (Led6, On);` ou `digitalWrite (Led6, Allume);` (les minuscules accentuées ne sont pas acceptées).

```
#define Led6 2
#define On LOW
#define Off HIGH
```

```
void setup () {
   pinMode (Led6,OUTPUT);
}
```

```
void loop () {
   digitalWrite (Led6, On);
}
```

Le programme est maintenant bien écrit. On peut le compléter pour clignoter la Led6. Charger `CliCentreA.ino`. Suivant les cas, d'autres LEDs peuvent être allumées. On n'a rien programmé les concernant.

**Clignoter un motif**

Sur le dé, allumons alternativement un 1 (LED au centre) et un 2 (2 LEDs en diagonale). On raisonne maintenant avec les motifs du dé, il faut pour chaque motif allumer plusieurs LEDs, et éteindre les autres. Cela fait 3 pins à définir et utiliser: Led6 et Led0+Led3

```
//Alterne12.ino
#define Led0 14
#define Led3 17
#define Led6 2

#define On LOW
#define Off HIGH
```

```
void setup () {
   pinMode (Led0,OUTPUT);
   pinMode (Led3,OUTPUT);
   pinMode (Led6,OUTPUT);
}
```

```
void loop () {
   digitalWrite (Led6, On);
   digitalWrite (Led0, Off);
   digitalWrite (Led3, Off);
   delay (1000) ; // 1 seconde
   digitalWrite (Led6, Off);
   digitalWrite (Led0, On);
   digitalWrite (Led3, On);
   delay (1000) ;
}
```

**Un petit chenillard**

On part de la Led4 pour aller en Led6 puis Led2 et revenir. Définissons toutes les LEDs, on pourra faire plusieurs programmes sans réfléchir au câblage.

```
//Cheni462.ino
#define Led0 14
#define Led1 15
#define Led2 16
#define Led3 17
#define Led4 18
#define Led5 19
#define Led6 2
#define Pous 3

#define On LOW
#define Off HIGH
```

```
void setup () {
   pinMode (Led0,OUTPUT);
   pinMode (Led1,OUTPUT);
   pinMode (Led2,OUTPUT);
   pinMode (Led3,OUTPUT);
   pinMode (Led4,OUTPUT);
   pinMode (Led5,OUTPUT);
   pinMode (Led6,OUTPUT);
   pinMode (Pous,INPUT);
}
```

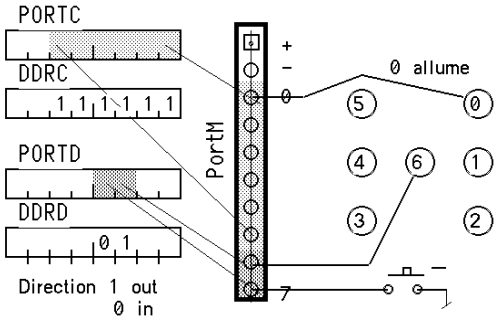
```
// oscille horizontalement
void loop () {
   digitalWrite (Led4,On);
   delay (500) ; // 1/2 seconde
   digitalWrite (Led4,Off);
   digitalWrite (Led6, On);
   delay (500) ;
   digitalWrite (Led6,Off);
   digitalWrite (Led1, On);
   delay (500) ;
   digitalWrite (Led1,Off);
   digitalWrite (Led6, On);
   delay (500) ;
   digitalWrite (Led6,Off);
}
```

Note: ce serait plus élégant de faire passer la dernière instruction en première position.
Pour être sûr que toutes les LEDs sont éteintes au démarrage, il faut ajouter dans le setup 7 instructions qui éteignent les LEDs. Vérifiez.

**Agir sur les registres et non pas sur les pins**

Un programme doit dire au processeur ce qui est entrée et sortie, c'est le ''setup'' qui avec Arduino définit des bits. Ici on va travailler directement avec les registres du processeur, comme on le fait dans toutes les applications industrielles du C.

Si nous nous intéressons à allumer des LEDs, elles peuvent être vues individuellement comme des pins Arduino. C'est souvent plus efficace de les voir comme un "vecteur", un mot de 8 bits que l'on retrouve sur le connecteur Kidule appelé PortM.
Les LEDs sont numérotées de 0 à 7 dans l'ordre des bits du PortK. Le tableau ci-contre donne la correspondance, et dans notre premier programme qui clignote la LED du centre, on met en évidence nos deux approches:




- Arduino – on utilise un digitalWrite (LedCentre,LOW); pour allumer
- C – on place sur le portM le mot binaire 0b10111111 qui active le bit 6, actif à zéro.

Comparons les deux approches. en clignotant la LED du centre

**Arduino**

```
//CliCentreA.ino
#define Led6 2
#define On LOW
#define Off HIGH

void setup() {
  pinMode(Led6,OUTPUT);
  digitalWrite (Led0, Off) ;
  digitalWrite (Led1, Off) ;
  digitalWrite (Led2, Off) ;
  digitalWrite (Led3, Off) ;
  digitalWrite (Led4, Off) ;
  digitalWrite (Led5, Off) ;
  digitalWrite (Led6, Off) ;

void loop() {
  digitalWrite (Led6, On) ;
  delay (500);
  digitalWrite (Led6, Off) ;
  delay (500);
}
```

**C**

```
//CliCentreC.ino
#define Led6On  0b10111111 // sur PortK
#define Led6Off 0b11111111

void setup () {
  DDRD |= 0b00001000 ;  // Led6 en sortie
  DDRD &= 0b0000010 ;  // Poussoir en entrée
  DDRC  = 0b00111111;   // Leds0-5 sortie, 6,7 inutilisé
  PORTC = 0b11111111;   // Leds éteintes
  PORTD|= 0b00000100; // Led 6 éteinte

}
void WriteKi (int kk) {
  PORTC = kk;
  PORTD != kk>>4;  // décalé de 4
  PORTD &= kk>>4;
}

void loop() {
  WriteKi (Led6On);
  delay (500);
  WriteKi (Led6Off);
  delay (500);
}
```

La version Arduino demande beaucoup de lignes pour éteindre toutes les LEDs.

En C, une ligne suffit pour éteindre les LEDs connectées au PORTC (état 1 partout).
Les notations qui agissent sur des mots binaires sont difficiles à bien comprendre, on les verra quand on aura un peu plus d'expérience. Acceptons-les telles quelles. Les définitions et le setup seront les mêmes pour les prochains exercices, pas besoin de modifier.
La fonction `WritePortM (mot binaire);` transfère cette valeur dans le kidule dé, via les Ports C et D du processeur. Les noms sont explicites, c'est ce qui compte. Nous ne connaissons pas encore assez la structure de l'Atmega328 et les instructions du C pour être à l'aise avec le setup. DDRD, DDRC sont des registres de direction, qui disent quels bits sont en entrée ou en sortie. Il y a beaucoup de registres de configuration dans le processeur, qui lui donnent sa flexibilité. Nous ne connaissons pas encore les opérations sur des mots binaires (signes &, | , <<) mais cela ne doit pas nous empêcher de progresser en utilisant des définitions et des fonctions qui marchent:

`#define LedCentreOn 0b10111111` envoyé sur le portM allume la Led6, éteint les autres
`WritePortM (etat);` envoie sur le PortM les 8 bits de la variable ou constante etat.

Donc si on veut former un 4 sur le dé, on définit
`#define Quatre 0b00101101` et pour afficher ce motif, on écrit dans le programme
`WritePortM (quatre);` ou
`WritePortM (valeur);` si on a calculé avant que `valeur=quatre;`

Pas convaincu par l'intérêt de travailler avec des mots binaires? Allumons une après l'autre les LEDs du dé (un chenillard). Avec Arduino, il faut tout déclarer, dire que 7 pins sont en sortie, les mettre à Off pour que l'affichage soit éteint au début. Et ensuite, il faut activer chaque ligne, attendre, désactiver, activer la suivante etc. Même un Nul fier d'être Nul se demande si on ne peut pas faire mieux!
En C, on raisonne avec une image du dé en mémoire et on utilise une procédure pour "balancer" cette image sur le dé. On définit les bits selon le câblage du dé, on agit sur les ports pour définir, 8 bits à la fois, la direction (input-output) et l'état initial. Ensuite, puisque qu'il faut activer les bits dans l'ordre où ils sont sur le registre, on décale et on teste la condition "tout décalé" pour recommencer.

**Arduino**

```
//CliSeqA.ino
#define Led0 14
#define Led1 15
#define Led2 16
#define Led3 17
#define Led4 18
#define Led5 19
#define Led6  2
#define PousDe 3
#define On LOW
#define Off HIGH

void setup() {
  pinMode(Led0,OUTPUT);
  pinMode(Led1,OUTPUT);
  pinMode(Led2,OUTPUT);
  pinMode(Led3,OUTPUT);
  pinMode(Led4,OUTPUT);
  pinMode(Led5,OUTPUT);
  pinMode(LedCentre,OUTPUT);
  digitalWrite (Led0, OFF);
  digitalWrite (Led1, OFF);
  digitalWrite (Led2, OFF);
  digitalWrite (Led3, OFF);
  digitalWrite (Led4, OFF);
  digitalWrite (Led5, OFF);
  digitalWrite (Led6, OFF);
}
void loop() {
  digitalWrite (Led0, ON); delay (500);
  digitalWrite (Led0, OFF);
  digitalWrite (Led1, ON); delay (500);
  digitalWrite (Led1, OFF);
  digitalWrite (Led2, ON); delay (500);
  digitalWrite (Led2, OFF);
  digitalWrite (Led3, ON); delay (500);
  digitalWrite (Led3, OFF);
  digitalWrite (Led4, ON); delay (500);
  digitalWrite (Led4, OFF);
  digitalWrite (Led5, ON); delay (500);
  digitalWrite (Led5, OFF);
```

**C**

```
//CliSeqC.ino
void setup () {
  DDRD &= 0b11110111 ;  // poussoir en entrée
  DDRD |= 0b00000100 ;  // led centre sortie
  DDRC |= 0b00111111;  // Leds 0-5
  PORTD |= 0b00000100;
  PORTC |= 0b00111111;  // Leds éteintes
}
void WriteKi (byte kk) {
  PORTC = kk;
  PORTD != kk>>4;  // décalé de 4
  PORTD &= kk>>4;
}
```

```
byte motif = 0b00000001 ;
void loop()  {
  WritePortM (~motif);
  delay (500);
  motif <<= 1 ;
  if (motif == 0b10000000) {
     motif=0b00000001 ;
  }
}
```

On raisonne avec un motif dit "logique" dans lequel un 1 allume, et au moment du transfert, on inverse (signe ~) pour être compatible avec la contrainte "physique" du câblage (Led allumée par un zéro).

Les opérations logiques sont présentées sous
http://www.commentcamarche.net/contents/c/cop.php3

```
  digitalWrite (Led6, ON); delay (500);
  digitalWrite (Led6, OFF);
}
```

**Exercices: inventez vos chenillards**

On a vu que `motif <<= 1 ;` décale le mot binaire à gauche. Des 0 sont injectés à droite.
`motif >>1 ;` décale à droite
Si on veut injecter un 1, on le force après décalage avec une addition: `motif+= 1 ;` ou `motif+= 0b10000000 ;` (plus court d'écrire `motif += 0x80 ;` )
Donc, avec quelques boucles `for` , des délais et des définitions de motifs, on peut créer des jeux lumineux amusants.

## Instruction `for (var;cond;modif)`

Pour répéter n fois un groupe d'instructions, on écrit
`for (int i=0 ; i<n ; i++) { groupe d'instructions}`
`int i=0` déclare une variable 16 bits initialisée à 0,
le groupe d'instructions s'exécute tant que `i<n`
`i++` est la modification à chaque répétition. ++ ajoute 1 à la variable `i.`

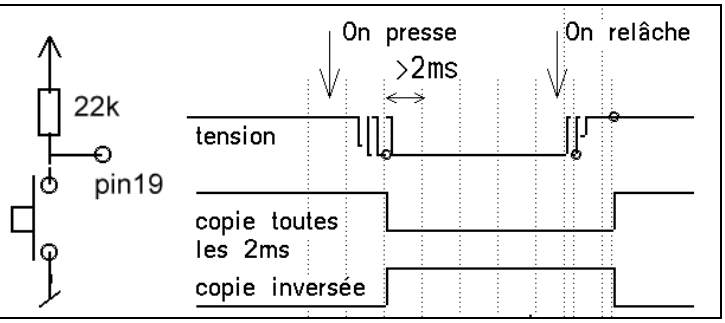
Comme exemple, clignotons 3 fois le centre, puis le pourtour.

```
// CliAlt.ino
//setup et fonction WrtitePortM () comme dans
l'exemple précédent
```

```
#define Centre   ~0b01000000
#define Pourtour ~0b00111111
void loop() {
  for (int i=0; i<3 ; i++) {
    WriteKi (~Centre);
    delay (200);
  }
  for (int i=0; i<3 ; i++) {
    WriteKi (~Pourtour);
    delay (200);
  }
}
```

## Le poussoir

Le poussoir est actif à zéro. Il faut déclarer la pin 19 comme une entrée avant de chercher à le lire.
`#define Pous 19`
`pinMode (Pous,INPUT) ;`
La lecture rend une valeur 0 ou 1, 0 actif
`digtalRead (Pous)`




Comme exemple simple, pour copier le poussoir sur la LED du centre, on écrit
`if (digitalRead (Pous) ==On) { digitalWrite (Led6,On); }`
`else {digitalWrite (Led6,Off);}`
Le poussoir est étudié avec plus de détails dans CKiFeux et CKiAf4.

## Instruction `while (condition)`

Le while (condition) permet de boucler on non selon une condition vrai/faux, donc une valeur différente de zéro ou égale à zéro .
Le groupe d'instructions entre { } est exécuté jusqu'à ce que la condition devienne fausse. Donc `while (1){}` boucle indéfiniment !
Avec le if, on fait si il faut et on continue. Avec le while, on fait tant qu'il faut.

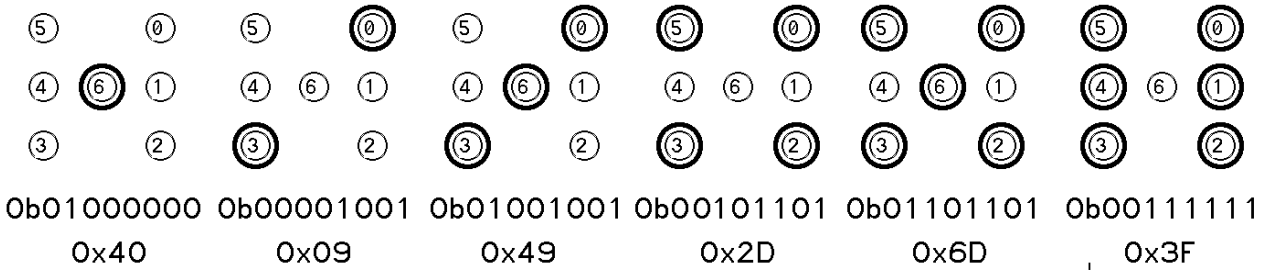
Pour copier le poussoir sur une LED, on peut aussi utiliser un while
`while ( digitalRead (Pous) ==On) { digitalWrite (Led6,On); }`
`digitalWrite (Led6,Off);`
On voit que si on presse, on répète sans cesse l'instruction d'allumage. Si on ne presse pas, on continue pour éteindre.

Si on veut agir quand on presse-relâche, le while est la seule solution

```
while ( digitalRead (Pous) ==Off)  {delay (20):}    // attend que l'on presse
while ( digitalRead (Pous) ==On)  {delay (20):}    // attend que l'on relâche
digitalWrite (Led6,!digitalRead (Led));              // inverse la Led
```

**Le dé**

Pour un dé électronique, nous avons 6 combinaisons d'allumage à créer. Avec Arduino, il faut chaque fois 7 instructions pour dire quelles LEDs on allume et éteint. On connaît leur position sur le connecteur du kidule dé, on va donc définir les 6 motifs en binaire.




Le programme `DeFaces.ino` affiche directement ces combinaisons et montre une autre façon de faire des chenillards.

```
//DeFaces.ino
void setup ()
{
  DDRD = 0b00000100;   // led centre sortie
  DDRC = 0b00111111;   // Leds 0-5
  PORTD |= 0b00000100;
  PORTC |= 0b00111111;  // Leds éteintes
}
void WritePortM (int mm)
{
  PORTC = mm ;
  PORTD = mm>>4 ;
}
#define Un 0x40
#define Deux 0x09
#define Trois 0x49
#define Quatre 0x2D
#define Cinq 0x6D
#define Six 0x3F
```

```
void loop()
{
  WritePortM (~Un);
  delay (500);
  WritePortM (~Deux);
  delay (500);
  WritePortM (~Trois);
  delay (500);
  WritePortM (~Quatre);
  delay (500);
  WritePortM (~Cinq);
  delay (500);
  WritePortM (~Six);
  delay (500);
}
```

De là, avec quelques `if` , on a un dé électronique. Il faut un compteur qui compte de 1 à 6 et s'arrête au hasard quand on pèse sur le poussoir pour demander une nouvelle valeur.
Pendant que l'on pèse, on augmente ce compteur spécial (après 6 on a 1) à toute vitesse, donc on ne peut pas deviner quand il faut relâcher pour piper le dé.
Le programme `DeJoue1.pde` est à tester, mais faisons mieux.

**Tableaux**

Un tableau est une collection de variables qui sont accessibles à l'aide d'un numéro d'ordre (index).
Attention, les 6 faces du dé, et leur motif de 0 et 1 qui définit l'allumage, sont numérotés de 0 à 5.
Le tableau se définit `byte Faces[6]={0x40,0x09,0x49,0x2D,0x6D,0x3F);`
La commande `Faces[6]` a rempli 6 positions mémoires successives et on peut retrouver le contenu de la 3e position en écrivant `Faces(2)` (2 et pas 3 car la 1ère est 0).
Pour afficher, c'est très simple puisque l'on peut écrire directement sur le PortM.
Jouez avec le programme `DeJoue2pde` et vérifiez sur 100 lancers que la probabilité d'apparition de chaque chiffre est correcte.
Quand on pèse sur le bouton, on voit toutes les LEDs allumées. En fait elles clignotent ; agitez le Kidule de droite à gauche, en rond, en zig-zag, c'est très joli !
Certaines LEDs du dé s'allument 1/6e du temps, d'autres ½ du temps, etc (vous comprenez pourquoi ?), Donc on voit des segments lumineux de longueur variable. Que se passe-il si on modifie le delay ?

DeJoue1.ino

```
...
byte counter = 1 ;
void loop()
{
  if (digitalRead(PousDe)=LOW)
  {
    counter++;
    if(counter==7)
    {
      counter=1;
    }
    delay (2) ;
  }
  else
  {
    if (counter==1)
     { WritePortM (~Un); }
    if (counter==2)
     { WritePortM (~Deux); }
    if (counter==3)
     { WritePortM (~Trois); }
    if (counter==4)
     { WritePortM (~Quatre); }
    if (counter==5)
     { WritePortM (~Cinq); }
    if (counter==6)
     { WritePortM (~Six); }
  }
}
```

DeJoue2.ino

```
….
byte counter = 1 ;
void loop()
{
  if (digitalRead(PousDe)==LOW)
  {
    counter++;
    if(counter==7)
    {
      counter=1;
    }
    delay (2) ;
  }
  else
  {
   WritePortM (~Faces[counter-
1]);
  }
}
```

Quand on pèse, le compteur avance, mais on ne le voit pas. Quand on lâche, il n'avance plus et on affiche la valeur pointée.

DeJoue3.ino

```
...
byte counter = 1 ;
void loop()
{
  if (digitalRead(PousDe)==LOW)
  {
    counter++;
    if(counter==7)
    {
      counter=1;
    }
    delay (2) ;
    WritePortM (~Faces[counter-1]);
  }
}
```

Le compteur est toujours affiché. On le voit bouger en agitant le dé. Si on augmente le délai, on peut tricher!
Le compteur compte de 1 à 6, mais la table est de 0 à 5. Il faut corriger l'index.

jdn/120411/131019/140127(180602