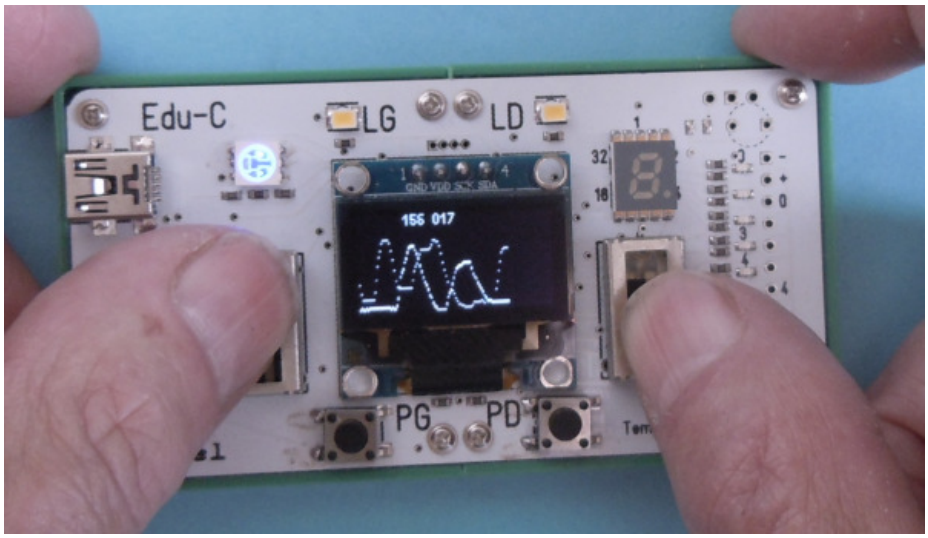
www.didel.com/educ/EduC-Fun.pdf

EduC – Découverte de la programmation

- sans apprendre à programmer ! –



Introduction

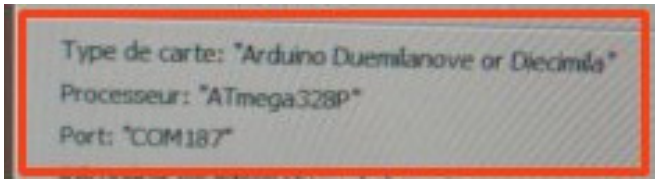
Edu-C a été développé pour apprendre à programmer en C, mais ici, on suit une approche ludique, en chargeant des programmes que l'on comprend avec un peu de bon sens, et on devine et teste ce qu'il faut faire pour modifier le programme. Des fonctions simples à utiliser ont été définies pour éviter des fonctions C plus générales, donc plus difficiles à utiliser correctement.

Pour apprendre le C avec Edu-C, un cours formé de 7 modules dont les liens sont accessibles via www.didel.com/EduC.htm permet de progresser. Le MOOC EPFL "Comprendre les Microcontrôleurs" est plus complet pour apprendre la programmation et la structure interne des microcontrôleurs.

EduC est livré avec 16 programmes de test et démonstrations. Mais ce n'est pas une console de jeux, c'est un outil pédagogique. Vous devez charger des programmes, les comprendre, les modifier. Intuitivement avec ce document Fun, ou systématiquement avec nos 7 modules ou d'autres cours.
Et si vous connaissez le C, améliorer ou compléter les librairies et créer des démos originales sur Edu-C vous offrira de quoi occuper vos soirées.

Installer le logiciel et télécharger les fichiers

Tester des programmes nécessite un environnement de programmation. Arduino est bien connu, et si vous n'avez pas l'habitude des PC et Mac, un ami vous aidera à installer.



Pour le développement, la carte est connectée à un PC ou Mac. Arduino et le driver CH340/WCH doivent être installés. Voir www.didel.com/educ/ArduinoInstall.pdf .
Si cette installation est faite, il faut charger l'ensemble des programmes de cette introduction sous www.didel.com/educ/Fun.zip
Il faut extraire le dossier (ne pas ouvrir) qui contient les dossiers Fun*xx* avec dans chacuns le programme Fun*xx*.ino et les librairies associées.

Edu-C est livré avec un programme qui permet de sélectionner 16 démos/programmes de test, documentés sous www.didel.com/educ/EduC-Demo.pdf.
C'est aussi une source d'exemples qui peuvent inspirer votre compréhension.

La carte est connectée, le dossier Fun est au coin de l'écran? Top départ!

1 Dessinons des montagnes – FunDraw.ino

Le résultat se voit dans la photo du haut: Edu-C a deux potentiomètres qui donnent des valeurs de 0 à 255. L'écran a 64 pixels de haut et 128 de large. On va mettre la valeur des pots verticalement et avancer horizontalement 20 fois par seconde, donc toutes les 50ms. Donc il faut diviser la valeur des pots par 4 pour avoir la distance verticale, et il faut encore savoir que tous les écrans informatiques balayent les pixels depuis le haut. Donc, si la valeur du pot est 0, c'est la valeur verticale 63 qu'il faut donner à l'écran. La "formule" pour passer du pot à l'écran est donc (63-pot/4). On a deux pots G et D, leurs noms sont connus par le fichier "Fun.h" qui sait tout sur Edu-C et "Oled.h" qui sait placer les pixels. Il faut quelques lignes au début pour que le compilateur Arduino mette tout en place et si toutes les () {} et ; sont bien placés, les minuscules et majuscules correctes, le compilateur pourra traduire en binaire, charger la mémoire et exécuter le programme.

Mais il faut d'abord amener le programme dans l'écran!

1) Ouvrir le dossier Fun préparé dans l'écran
2) Cliquer sur le programme FunDraw
3) Cliquer sur FunDraw.ino – Arduino s'ouvre et le texte apparaît
4) Cliquer sur la 2e flèche, le programme se compile est se charge (la Led verte clignote pendant le chargement). S'il y erreur de chargement, vérifier sous "outils" que les paramètres d'installation sont toujours corrects.

Le programme tourne?
Il est facile à comprendre: on exécute sans cesse après le **`void loop()`** les instructions exécutées entre accolades **`{ }`**.
On a défini deux variables, **`potG`** et **`potD`** qui sont des "enveloppes" pour transporter les valeurs que l'on lit sur les pots avec les fonctions **`GetPotG()`** et **`GetPotD()`.**
Les fonctions graphiques **`Dot(x,y);`** et **`DDot(x,y);`** (2 points superposés) prennent ces valeurs et la valeur x augmente chaque fois. Il y a 64 pixels verticalement et les pots donnent des valeurs de 0 à 256; on doit donc diviser par 4. L'origine de l'écran est en haut.
Pressez le poussoir gauche pour recommencer. OK?

```
//FunDraw.ino
#include "Fun.h"
#include "Oled.h"
void setup(){ SetupFun(); SetupOled(); }

byte potG,potD;  //x y sont déclarés dans Fun.h
void  loop() {
   potG = GetPotG(); potD = GetPotD();
   Dot (x,63-potG/4); DDot (x,63-potD/4);
   if (x++>=128) {x=0; AttPousG(); Clear(); }
   DelMs(50);
}
```

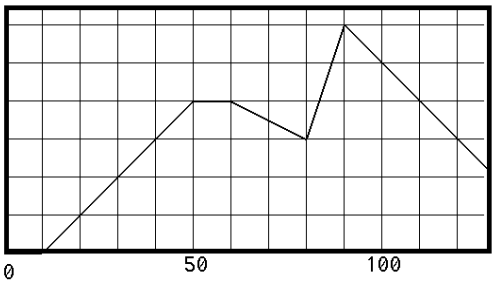
Comprenons le programme. Les deux premières lignes de la boucle sont assez claires. Il faut les modifier, prévoir l'effet de la modification et vérifier. On change quelque part, on vérifie que la syntaxe est toujours correcte, on prévoit le résultat, on exécute et on voit si on a bien réfléchi !
La 4e ligne DelMs(50); attend 50ms, donc définit la vitesse horizontale. Le paramètre dans la parenthèse peut valoir de 1 à 65535 (limite d'un mot de 16 bits).
La 3e ligne est intéressante à comprendre. x++ dit que l'on augmente x de un. Une décision est prise juste avant: si x est supérieur ou égal à 128, on fait ce qui est regroupé dans l'accolade: on remet le pointeur x au début de la ligne et on efface l'écran. On pourrait recommencer tout de suite, mais c'est mieux d'avoir le temps d'admirer son œuvre. Avec la fonction AttPousG(); on attend une pression sur le poussoir gauche pour continuer. C'est fait avec quelques instructions en C mises dans la libraire Fun.
Une autre solution serait de mettre un DelMs(1000); et cela recommencerait après une seconde.
Bon conseil: Sauvez vos programmes modifiés en leur donnant des noms: Draw1.ino, Draw2.ino par exemple.

Montagard -Un premier jeu d'adresse

Dessinons une montagne, et avec le potentiomètre, il faudra la suivre, et dire si on a bien fait.
Premier problème, et premier programme de test: dessiner la montagne avec des **`Dot(x,y);`**. Pour une horizontale, on augmente x avec y constant, pour une verticale …vous devinez. Pour une oblique, à 45° c'est facile. Pour une autre pente, il faut multiplier par des nombres fractionnaires, arrondir, cela suppose de savoir un certain nombre de choses en programmation C, donc on va rester à 45 degrés, et 2 fois plus raide ou 2 fois moins raide (vous devinez comment faire?).

Dessinons la montagne sur du papier quadrillé. On part du bord de mer, pot à zéro. En 20, 60, 70, 90 on change la pente et on s'arrête à 127 (que se passe-t-il si on va trop loin?).

```
void loop() {
   x=0; y=0;
   Repete (10) {DDot (x++,63-0);}          // plateau
   Repete (40) {DDot(x++,63-y++);}         // 45 degrés
   Repete (10) {DDot(x++,63-y);}           // plateau
   Repete (20) {DDot(x++,63-y--);x++;}     // petite descente
   Repete (5)  {DDot(x++,63-y);}           // plateau
   Repete (10) {DDot(x++,63-y); y+=3;}     // montée raide
   Repete (33) {DDot(x++,63-y--);}         // -45 degrés
}     // total 128 (on est parti de zéro)
```



Que veut dire `y+=3; ? --> y=y+3;` C'est un truc du C, comme le y++ qui remplace y=y+1.
Le programme complet s'appelle **FunMontagne.ino**. Chargez-le depuis le dossier Fun et faites vos variantes de montagnes **FunMontagne1.ino FunMontagne 2.ino** etc.

`Repete(nfois){}` et **`Stop();`** ne sont pas des instructions du C, mais des fonctions très faciles à écrire en C; pas besoin d'expliquer ce qu'elles font. Mais Stop(); ne stoppe rien du tout! Le processeur tourne dans une boucle où il ne fait rien – 10 millions de fois par seconde. Le bouton reset redémarre au début, qui n'est pas le début de votre programme, il faut vérifier la communication USB avec le PC avant.

```
//FunMontagne.ino
#include "Fun.h"
#include "Oled.h"
void setup(){ SetupFun(); SetupOled(); }
void loop() {
   x=0; y=0;
   Repete (10) {DDot (x++,63-y);}         // plateau
   Repete (40) {DDot(x++,63-y++);}        // 45 degrés
   Repete (10) {DDot(x++,63-y);}          // plateau
   Repete (20) {DDot(x++,63-y); y=((2*y)-1)/2;}  // petite descente
   Repete  (5) {DDot(x++,63-y);}          // plateau
   Repete (10) {DDot(x++,63-y); y+=3;}    // montée raide
   Repete (33) {DDot(x++,63-y--);}        // -45 degrés
 // total 127 (on est parti de zéro)
   Stop();
}
```

Qui est le champion?

Le but est de suivre la montagne au mieux. Pour chaque valeur de x, il faut regarder la différence entre la montagne, la valeur y que l'on doit recalculer parce que l'on ne peut pas lire ce qu'il y a sur l'écran, avec la valeur **`GetPotG()/4`**. Comme ce **`GetPotG()/4`** est un peu embêtant à écrire, et qu'on va le voir très souvent, ont peut définir le synonyme **`Gpot`** en écrivant

```
#define Gpot (GetPotG()/4)     // pas de ; à la fin
```

Que faire si le montagnard passe à travers la montagne? On peut décider que c'est raté et on recommence, on peut soustraire la valeur, on peut ne pas soustraire mais ajouter des points négatifs, on peut ignorer. Le plus simple est de totaliser la valeur absolue des écarts. Il y a une fonction C pour cela: **`abs(a,b);`** calcule **`a-b`** si **`a>=b`** et **`b-a`** si **`a<b`**.

Le programme commence par dessiner la montagne, mais l'écran ne peut pas être lu pour savoir où se trouve la montagne, et ce serait compliqué. Le plus simple est de la redessiner, tout en affichant un **`Dot()`** selon le potentiomètre et en effectuant le calcul de l'erreur. Donc pour le premier trajet:

```
Repete (10) {DDot (x++,63-y); Dot(x,63-Gpot; tot += abs(Gpot-y); Del;}
```
plateau, y ne change pas — la position du pot — l'erreur — un délai

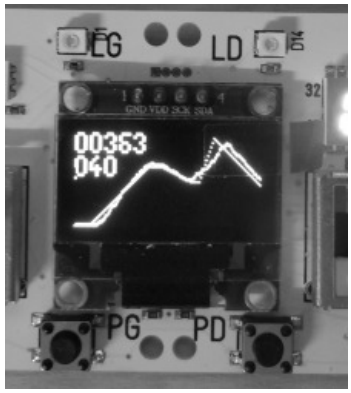
`Del;` est aussi un synonyme, une définition qui nous permettra de changer ce délai à un seul endroit au début, et la mise à jour se fera dans toutes les lignes. Avec un délai de 100ms, on suit assez facilement. Dans la définition de Del, on rajoute l'affichage du score. On a aussi décidé que le délai est une variable appelée `dl`, initialisée à 50ms, et qui diminue de 5 à chaque partie. Un décompteur prépare aussi le départ du trek. Il utilise la fonction **`Digit(n);`** dont le paramètre est le chiffre décimal ou hexadécimal affiché. Le jeu recommence en accélérant le déplacement, donc en diminuant le délai de 50 à 5 ms.

```
//FunTreck.ino
#include "Fun.h"
#include "Oled.h"
```

```
void setup(){ SetupFun(); SetupOled();  }
#define Gpot (GetPotG()/4)
int dl = 50;  // variable on augmentera la vitesse a 5 on recommence
#define Del DelMs(dl); LiCol(1,0);BigDec16(tot);

void loop() {
   Clear();  x=0; y=0;
   Repete (10) {Dot (x++,63-0);}        // plateau
   Repete (40) {Dot(x++,63-y++);}       // 45 degrés
   Repete (10) {Dot(x++,63-y);}         // plateau
   Repete (20) {Dot(x++,63-y); if(x%2){y--;}  }  // petite descente
   Repete (5)  {Dot(x++,63-y);}         // plateau
   Repete (10) {Dot(x++,63-y); y+=3;}   // montée raide
   Repete (33) {Dot(x++,63-y--);}       // -45 degrés
   // total 128 (on est parti de zéro)

   // Attention départ
   Digit(3);DelMs(1000);Digit(2);DelMs(1000);Digit(1);DelMs(1000);Digit(0);
    int tot=0; // total des mauvais points  tot+=x; est comme tot=tot+x;
   x=0; y=0;
   Repete (10) {DDot(x,63-Gpot); Dot (x++,63-y);  tot += abs((Gpot)-y); Del;}
   Repete (40) {DDot(x,63-Gpot); Dot(x++,63-y++); tot += abs((Gpot)-y); Del;}
   Repete (10) {DDot(x,63-Gpot); Dot(x++,63-y);   tot += abs((Gpot)-y); Del;}
   Repete (20) {DDot(x,63-Gpot); Dot(x++,63-y); if(x%2){y--;} tot += abs((Gpot)-y); Del;}
   Repete (5)  {DDot(x,63-Gpot); Dot(x++,63-y);   tot += abs((Gpot)-y); Del;}
   Repete (10) {DDot(x,63-Gpot); Dot(x++,63-y); y+=3; tot += abs((Gpot)-y); Del;}
   Repete (33) {DDot(x,63-Gpot); Dot(x++,63-y--); tot += abs((Gpot)-y); Del;}
   dl -=5; if (dl==0) dl=50;   // accélère
   Attend(PousG);
}
```



Note: Si vous connaisez le C, `Repete(nn){}` est la sténo pour `for(byte i=0, i<nn; i++) {}`

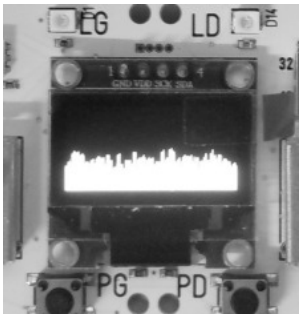
Les nombres aléatoires

Arduino propose la fonction **`random (Min,Max)`**; qui rend un nombre entre Min et Max-1.
Ce nombre est calculé, il suit la même séquence si on ne change pas un paramètre initial avec la fonction randomSeed(valeur). Pour plus de détails, voir https://en.wikipedia.org/wiki/Random_seed et les exemples simples dans www.didel.com/EduMod3.pdf.

Tirons des nombres au hasard, de 0 à 127. Comptons combien de fois chaque nombre apparaît. Et affichons le résultat sur le Oled sous forme d'un histogramme. Construisons dans la position x=13 un segment vertical dont la longueur est égale au nombre de fois où le 13 a été tiré. Il nous faut donc 127 compteurs, regroupés dans une table définie par: **`byte taHisto[128];`** A chaque tirage, on marque le point correspondant sur l'écran.

```
//FunOledRandom.ino  essai vecteur horiz
#include "Fun.h"
#include "Oled.h"
void setup () { SetupFun(); SetupOled(); randomSeed(GetPotD()); }

byte taHisto[128];
byte mesure,yy,moy;
int cnt;
void loop() {
  mesure = random (0,128);
  yy = ++taHisto[mesure];
  DDot (mesure,63-yy);
  if (yy==63) {while(1);} //fini
  //  delay (10);
}
```

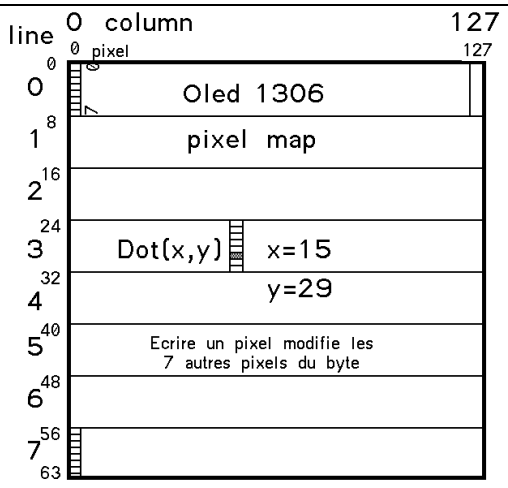


Afficher des nombres et des textes

Si vous connaissez le "terminal" Arduino, on retrouve les mêmes facilités, avec l'avantage d'un écran fixe, mais évidemment il faut bien dire chaque fois dans quelle ligne et à partir de quelle colonne on écrit.
Les gros caractères (Big..) prennent 2 lignes de haut et ne passent pas en ligne 0.
`LiCol(li,col)` avec `li` valant 0 à 7 et `col` 0 à 127 positionnent le pointeur. Les fonctions sont

```
Car('a');  Text("txt");  Sprite(smile);
Bin8(v8); Hex8(v8); Hex16(v16);
Dec8(v8); Dec16(v16); Dec9999(v13);
Big('a');  BigBin8(); BigHex8(); BigHex16();
BigDec8();  BigDec9999();

Dot(x,y); DDot(x,y); Vline(x); Hline(y);
```



Lutins

Un lutin (sprite) doit être défini dans une table. Cette table est en librairie pour les sprites Smile,et Sad, appelés par `Sprite(Smile);`
Si le sprite est dans la mémoire programme de l'utilisateur, comme la croix ci-contre, il faut déclarer

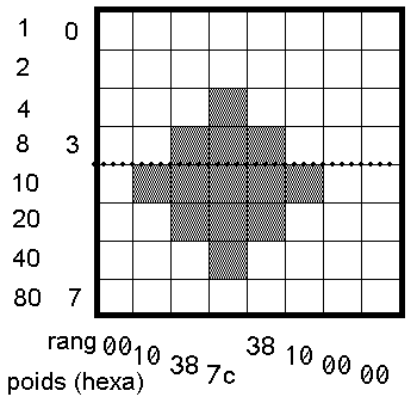
```
byte croix[]= {0x00,0x10,0x38,0x7c,0x38,0x10,0x00,0x00};
```

L'instruction pour copier au centre de l'écran est

```
LiCol(3,60); MySprite[croix];
```

La longueur est évidemment inférieure à 127.

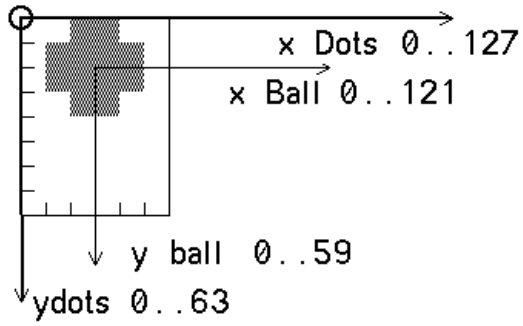
Pour un lutin de 16 de haut, il faut déclarer 2 sprites de 8 de haut et les aligner avec deux `LiCol(,);`.
On ne peut pas déplacer facilement un lutin. Pour l'effacer, in faut définir un lutin vide de même dimension.



Balle de 4x4

La fonction Ball (x,y) est un objet de 4x4 pixels dans une champ de 6x6 pour que le déplacement d'une case dans toutes les directions efface la balle précédente. Toutefois, si le déplacement est rapide, le soft demandera peut-être un déplacement de plus d'une unité, qui laissera un caca.

Le point de référence est au centre de la balle, donc on peut la déplacer selon 122x60 positions.
La fonction `Ball(x,y);` positionne la balle sur le terrain et sature à 121/59. Elle utilise les variables globales x,y (déclarées dans Oled.h ou Ping.h).



Les fonctions pour déplacer la balle sont délicates, et il y a inévitablement des cas de recouvrement. Les librairies GFX évitent ces recouvrements, mais ne sont pas assez rapides pour jouer au ping-pong.

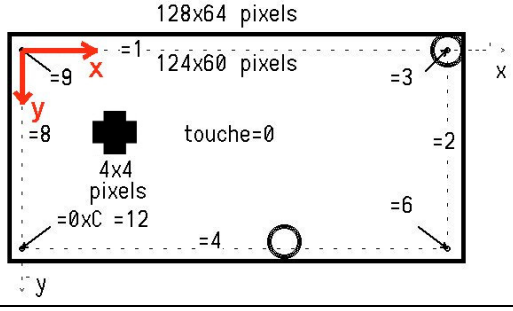
Test de déplacement

La balle est déplacée avec les potentiomètres G (horizontal) et D (vertical). Les valeurs ne sont pas saturées; facile de rajouter des `if(x>122) {y=122;},` ce sera nécessaire.
Un cadre est construit avec une balle dans chaque coin. On peut vérifier comment le cadre est démoli si on ne sature pas. On voit aussi qu'il faut aller lentement, pour que la balle se déplace de 1 à la fois.

```
//FunPotBall.ino
//On bouge la balle au pot et on affiche la variable touch
#include "Fun.h"
#include "Oled.h"
void setup () { SetupFun(); SetupOled(); }

void loop() {   // test trajectoire
  Hline(0);   Hline(63); Vline(0); Vline(127);
  Ball(0,0); Ball(122 ,0); Ball  (122,57); Ball(0,57);
  while(1){
    x=GetPotG()/2; y=GetPotD()/4;
    Ball (x,y); Touche();
    LiCol(5,50);Hex8(x); Hex8(y); Bin8(touch);
    DelMs(1);  // pour ralentir
  }
}
```

Une fonction très intéressante de la librairie est testée par ce programme. La fonction `Touche();` met à jour la variable globale `touch` qui vaut zéro au centre de l'écran, et prend les valeurs indiquées sur le dessin quand la balle touche un bord ou un coin.

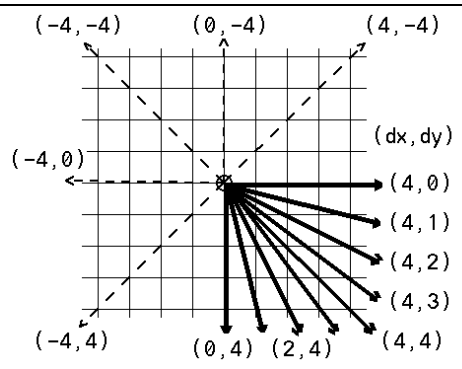


Jouer avec des balles

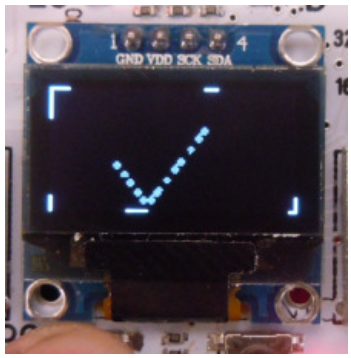
On peut multiplier les balles sur l'écran, si elles ne se superposent pas. On peut les déplacer de +/-1 à la fois en x et y, Imaginez des jeux qui pilotent des **`Ball(x,y) ;`** et **`Dot(x,y);`** .

La balle se déplace seule

La fonction **`PosDir (x,y, dx,dy);`** définit la position initiale de la balle et sa direction, avec pour dx dy des valeurs de -4 à +4, donc seulement quelques inclinaisons possibles.
La fonction **`Step();`** déplace la balle d'un cran selon dx/dy et met à jour la variable globale prédéfinie **`touch`** qui prend l'une des 10 valeurs selon le bord touché.
La vitesse de la balle dépend de la durée entre deux **`Step();`** Cette durée est la somme des actions à entreprendre: le dessin de la balle est assez long (2ms) et on le complète par un délai pendant la mise au point.



Quelle décision prendre quand la balle touche un bord?
La balle va rebondir contre une paroi verticale si on pose **`dx=-dx;`** .
Elle revient sur elle-même si on a { **`dx=-dx; dy=-dy; }`**
Le programme suivant rebondit en détruisant les bords:



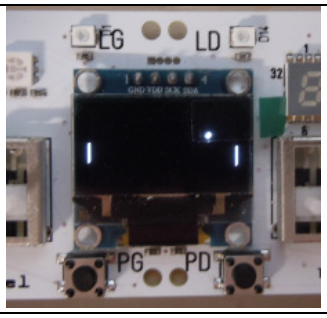

```
//FunBall.ino  2320 b 137v
#include "Fun.h"
#include "Oled.h"
void setup () { SetupFun(); SetupOled(); }

 void loop() {   // test trajectoire
  Hline(0); Hline(63); Vline(0); Vline(127);
  Ball(0,0); Ball(122 ,0); Ball  (122,57); Ball(0,57);
  while(1){
    PosDir(64,32,3,4);
    while(1) {
      Step(); DelMs(1);
      if (touch&1) {dy=-dy;}
      if (touch&2) {dx=-dx;}
      if (touch&4) {dy=-dy;}
      if (touch&8) {dx=-dx;}
      while (!PousG) ;
    }
  }
}
```

Le codage binaire utilisé pour touch est astucieux. Si on est par exemple dans le coin 9 = 8+1, deux conditions `if` sont satisfaites et la balle revient sur elle-même, ce qui est bien ce que l'on veut. Il n'y a pas besoin de prévoir ces 4 cas.

Raquette

Pour des segments horizontaux ou verticaux, on utilise Dot(x,y) dans une boucle **`Repete(nn);`** (ou boucle **`for`**).
Les raquettes ont une contrainte difficile: écrire une ligne verticale prend du temps et l'option qui minimise ce temps laisse des cacas si on se déplace très rapidement.
Pour le ping-pong deux raquettes sont programmées en x1=0 et x2=127.
La variable **`touch`** est mise à jour dans ce cas, et il faut utiliser les variables globales y1 et y2 pour positionner les 2 raquettes.
y1 et y2 dépendent des potentiomètres PotG et PotD.



La raquette a 16 pixels de haut et son point haut se déplace de 0 à 48. La formule (3x(256-Pot)/16 donne une valeur de 0 à 48 avec la valeur 0 quand le pot est en haut.
Le dessin de la raquette dure 3ms. Le dessin de la balle est ~2ms. Le temps minimum pour traverser l'écran est de 127x8ms ~= 1 seconde.

Si la balle touche les raquettes ainsi définies, la variable touch voit les bits 4 et 5 (valeur 0x10=16 et 0x20=32) activés. On a tout ce qu'il faut maintenant pour programmer le ping-pong.

```
//FunPingPong.ino
#include "Fun.h"
#include "Oled.h" // 17h fast
void setup () { SetupFun(); SetupOled(); }

byte v1,v2; // vitesse de la raquette
byte oy1,oy2;
void loop() {   // rebondit raq
    PosDir(64,32,4,1);
    while(1) {
 //    DelMs(10); //pour ralentir l'aller-retour en 2s
      Step();
      y1 = ((256-GetPotG())*3)/16;
      Raq (0,y1);
      y2 = ((256-GetPotD())*3)/16;
      Raq (127,y2);
      v1= abs(oy1-y1); v2= abs(oy2-y2);
      oy1=v1; oy2=v2;
      Touche();
      if (touch&1) {dy=-dy;}
      if (touch&2) {goto ko;}
      if (touch&4) {dy=-dy;}
      if (touch&8) {goto ko;}
      if (touch&16) {dx=-dx; }
      if (touch&32) {dx=-dx;}
      //while (!PousG) ;
   }
ko:
  LiCol(3,50); BigText("KO");LiCol(5,20); Text("PousD recommence");
  while(!PousD);
  Clear();
  nop;
}
```

On voit que la "vitesse de déplacement des pots" v1 et v2 est calculée. La valeur semble être entre 0 et 5. L'idée est de couper les balles, et agir sur l'angle de renvoi et/ou la vitesse.
Plus simple à programmer avec ce que l'on a vu, on peut attendre au démarrage sur un poussoir et modifier la vitesse ou l'angle de départ. Améliorer les textes, afficher un score est facile. Mais pendant que la balle bouge, le moindre affichage va ralentir.

Autres jeux?

Il y a des programmes simples de réflexes, comme les programmes 7 et 10 de la démo.
On peut jouer avec des shows lumineux, des nombres aléatoires, programmer le jeu de Simon et on trouve beaucoup d'idées sur le Web.

Pour un casse-briques et autres programmes sur écran. il y a soit des contraintes de superposition soit de vitesse . Développer les primitives comme celles nécessaires pour notre ping-pong nécessite beaucoup de réflexion et d'optimisations.

Note concernant la librairie "Fun.h"

Les fonctions déclarées contiennent en plus des fonctionnalités propres à la carte Edu-C (librairie EduC.h) des fonctions qui remplacent des instructions C et simplifient la compréhension des programmes.
La librairie Oled.h graphique a évolué en cours de développement et n'est pas stabilisée. Chaque programme contient les librairies qu'il utilise; elles n'ont pas besoin d'être identiques.

```
Repete(nn) {instructions};  identique à  for(byte i=0;i<nn;i++}
Boucle()  identique à  while(1) {instructions}
Stop();  identique à  while(1); Attend(cond) identique à while(!cond);
AttPousG();  AttPousD();  AttPousG(){faire en attendant}   Attente pression
```

A savoir si vous n'avez jamais programmé en C

Les minuscules et majuscules sont importantes.
Les instructions se terminent par un ;

// annonce un commentaire jusqu'à la fin de la ligne
les variables sont déclarées comme byte ou int.
Les paramètres constants et variables sont entre parenthèses. La parenthèse est vide s'il n'y en a pas.
Les conditions vrai/faux sont entre parenthèses (vrai?)
Les groupes d'instructions sont entre accolades {faire}
On peut mettre plusieurs instructions par ligne, si cela facilite la lecture.
On doit décaler et aligner les groupes d'instructions.
Pour faciliter le calcul, v++ prend la valeur v et ajoute un pour plus tard.
Les deux premières lignes seront toujours les mêmes (contraintes du C/Arduino).
De même, les instructions sont dans la structure "loop" d'Arduino; loop() {instructions}
Les variables et fonctions doivent être définies avant d'être utilisées.

Le programme `FunLum.ino` dans le dossier `FunLum` fait trois passes sur les Leds en tout ou rien, puis sur ces même leds en proportionnel, en utilisant aussi la fonction `Repete`, dérivée de la boucle for du C.
En fin de programme, on attend une pression sur le PousG pour recommencer.
Trois délais d'évolution en millisecondes sont déclarés, on peut les changer pour modifier le rythme.

```
//FunLum.ino Petit show
#include "Fun"
void setup(){ SetupFun();}

#define Dc DelMs(20)    // court
#define Dn DelMs(300)3 sec   // normal
#define Dl DelMs(1000)  // lent

void  loop();
  Repete(3) {LedGOn; Dn; LedDOn; Dn; LedGOff; Dn; LedDOff; Dn;}
  byte v5=0;
  Repete(32){LedG(v5++);Dc;}  Repete(32){LedG(--v5);Dc;}
  Repete(32){LedD(v5++);Dc;}  Repete(32){LedD(--v5);Dc;}
  Repete(32){Rouge(v5++);Dc;} Repete(32){Rouge (--v5);Dc;}
  Repete(32){Vert(v5++);Dc;}  Repete(32){Vert(--v5);Dc;}
  Repete(32){Bleu(v5++);Dc;}  Repete(32){Bleu(--v5);Dc;}
  AttPousG);
}
```

Table de mixage - Programme FunMix

Les deux potentiomètres donnent une valeur 8 bits (de 0 à 255). On peut utiliser pour mélanger les couleurs. Comme il n'y a que deux potentiomètres, on va utiliser les 2 poussoirs pour choisir sur quelle couleur agir

PousG pressé	PousD pressé
PotG -> Rouge PotD -> Vert	PotG -> Bleu PotD -> Intensité

Pour choisir le poussoir, on utilise l'instruction if: `if(PousG) {faire ce qu'il faut si pressé}`.
GetPotG() et GetPotD() donnent la valeur 8 bits des potentiomètres. Pour avoir une valeur de 0 à 31, il faut diviser par 8 (8*32=256).

```
//FunMix.ino On mélange les couleurs
#include "Fun.h"
#include "Oled.h"
void setup(){ SetupFun(); SetupOled();}
byte rr,vv,bb,ii;
void loop() {
  if(PousG) { rr=GetPotG()/8; vv=GetPotD()/8; }
  if(PousD) { bb=GetPotG()/8; ii=GetPotD()/8; }
  Rouge(rr); Vert(vv); Bleu(bb);
  LiCol(0,0); Text(" R   G   B  ii");
  LiCol(2,0);Dec8(rr); Dec8(vv); Dec8(bb); Dec8(ii);
}
```

Si on veut tenir compte de l'intensité, la valeur ii vaut de 0 à 31. Il faut donc multiplier les 3 couleurs par ii/32, donc définir de nouvelles variables.

```
//FunMixInt.ino. . .
byte rr,vv,bb,ii,ri,vi,bi;
void loop() {
  . . .
  ri=rr*ii/32; vi=vv*ii/32; bi=bb*ii/32;
  Rouge(ri); Vert(vi); Bleu(bi);
  LiCol(0,0); Text(" R   G   B  ii");
  LiCol(2,0); Dec8(ri); Dec8(vi); Dec8(bi); Dec8(ii);
  DelMs(100);

}
```