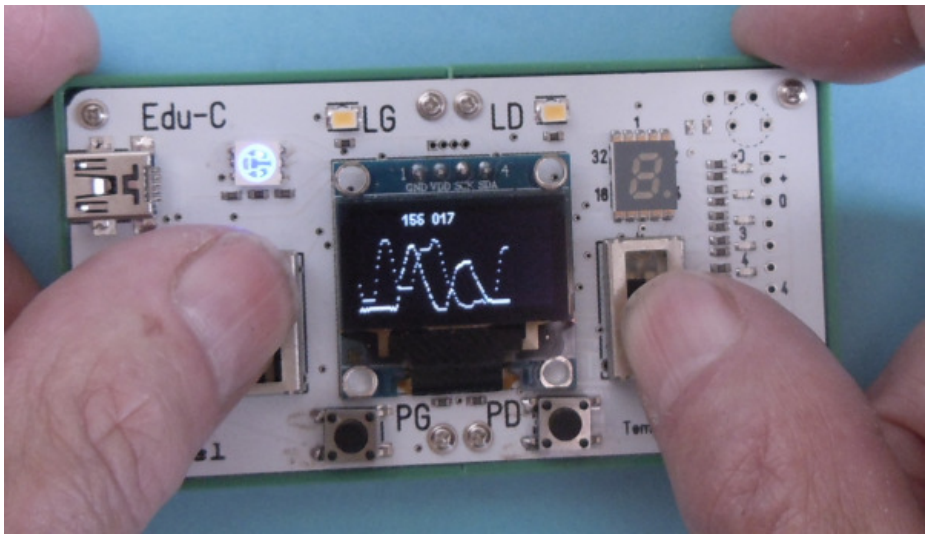
EduC - Discovery of programming
- learning by doing ! -



Introduction

Edu-C was developed to learn to program in C, but here we follow a fun approach, loading programs that we understand with a little common sense, and we guess and test what to do for modify the program. Simple to use functions have been defined to avoid more general C functions, which are more difficult to use correctly.

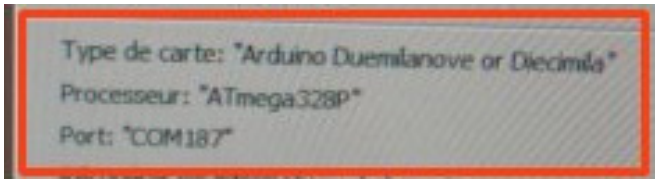
To learn C with Edu-C, a course (in French) consisting of 7 modules whose links are accessible via www.didel.com/educ/EduC.html allows progress. The EPFL MOOC EPFL "Comprendre les Microcontrôleurs" is more comprehensive for learning the programming and internal structure of microcontrollers.

EduC comes with 16 test programs and demonstrations. But it's not a game console, it's a teaching tool. You must load programs, understand them, modify them. Intuitively with this Fun document, or systematically with our 7 modules or other courses.
And if you know the C, improve or supplement the bookstores and create original demos on Edu-C will offer you something to occupy your evenings.

Install the software and download the files

Testing programs requires a programming environment. Arduino is well known, and if you are not used to PCs and Macs, a friend will help you install.é

For development, the card is connected to a PC or Mac. Arduino and the CH340 / WCH driver must be installed. See www.didel.com/educ/ArduinoInstall.pdf



If this installation is done, we must load all the programs of this introduction under www.didel.com/educ/Fun.zip

It is necessary to extract the file (not to open) which contains the Funxx folders with in each program Funxx.ino and the associated libraries.
Edu-C comes with a program that allows you to select 16 demos / test programs, documented sous www.didel.com/educ/EduC-Demo.pdf.
It is also a source of examples that can inspire your understanding.

The board is connected, the Fun folder is at the corner of the screen? Starting signal!

1 Drawing mountains - FunDraw.ino

The result can be seen in the picture above: Edu-C has two knobs that give values from 0 to 255. The screen is 64 pixels high and 128 wide. We will put the value of the pots vertically and advance horizontally 20 times per second, so every 50ms. So you have to divide the value of the pots by 4 to have the vertical distance, and you have to know that all the computer screens scan the pixels from the top. So, if the value of the pot is 0, it is the vertical value 63 that must be given to the screen. The "formula" for moving from the pot to the screen is therefore (63-pot / 4). We have two pots G and D, their names are known by the file "Fun.h" who knows everything about Edu-C and "Oled.h" who knows how to place the pixels. It takes a few lines at the beginning for the Arduino compiler to put everything in place and if all the () {} and; are well placed, the correct upper and lower case letters, the compiler can translate into binary, load memory and run the program.
But first you have to bring the program on the screen!
1) Open the prepared Fun folder on the screen
2) Click on the program FunDraw
3) Click on FunDraw.ino - Arduino opens and the text appears
4) Click on the 2nd arrow, the program is compiled and is loaded (the green LED flashes during loading). If there is a loading error, check under "tools" that the installation parameters are still correct.

The program is running?
It is easy to understand: we execute constantly after the void loop () the instructions executed in braces {}. We defined two variables, potG and potD which are "envelopes" to carry the values that we read on the pots with the functions GetPotG () and GetPotD ().
Graphics functions Dot (x, y); and DDot (x, y); (2 superimposed points) take these values and the value x increases each time. There are 64 pixels vertically and pots give values from 0 to 256; we must divide by 4. The origin of the screen is at the top.
Press the left pusher to start again. OK?

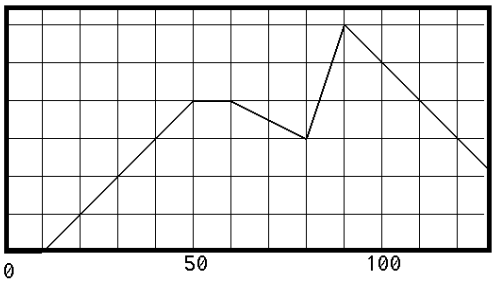
```
//FunDraw.ino
#include "Fun.h"
#include "Oled.h"
void setup(){ SetupFun(); SetupOled(); }

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);
}
```

Let's understand the program. The first two lines of the loop are pretty clear. You have to modify them, predict the effect of the modification and check. We change somewhere, we check that the syntax is always correct, we predict the result, we run and we see if we thought well!
The 4th line DelMs (50); wait 50ms, so sets the horizontal speed. The parameter in the parenthesis can be from 1 to 32767 (limit of a 16-bit signed word).
The 3rd line is interesting to understand. x++ says that we increase x by one. A decision is made just before: if x is greater than or equal to 128, we do what is grouped in the brace: we put the pointer x at the beginning of the line and we clear the screen. We could start right away, but it's better to have the time to admire the work. With the AttPousG () function; you wait for a pressure on the left pusher to continue. It's done with some instructions in C put in the bookseller Fun.
Another solution would be to put an DelMs (1000); and it would start again after a second.
Good advice: Save your modified programs by giving them names: Draw1.ino, Draw2.ino for example.

Montagard -A first game of skill

Let's draw a mountain, and with the potentiometer, it will be necessary to follow it, and to say if one did well. First problem, and first test program: draw the mountain with Dot (x, y); For a horizontal, we increase x with constant y, for a vertical ... you guess. For an oblique, 45 ° is easy. For another slope, it is necessary to multiply by fractional numbers, to round, that supposes to know a certain number of things in programming C, so we will stay at 45 degrees, and 2 times steeper or 2 times less steep (you guess how make?).
Let's draw the mountain on squared paper. We start from the seaside, pot to zero. In 20, 60, 70, 90 we change the slope and we stop at 127 (what happens if we go too far?).

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



What does `y+=3; ? --> y=y+3;` mean? It's a trick of C, like yy ++ replacing y=y+1.
The complete program is called **FunMontagne.ino**. Load it from the Fun folder and make your Mountain variants **FunMontagne1.ino FunMontagne 2.ino** etc.

Repeat (ntimes) {} and Stop (); are not instructions of the C, but functions very easy to write in C; no need to explain what they do. But Stop (); do not stop anything at all! The processor runs in a loop where it does nothing - 10 million times per second. The reset button restarts at the beginning, which is not the beginning of your program, you have to check the USB communication with the front PC.

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

Who is the champion?

The goal is to follow the mountain at best. For each value of x, we must look at the difference between the mountain, the value y that we must recalculate because we can not read what is on the screen, with the value GetPotG () / 4. As this GetPotG () / 4 is a bit annoying to write, and we will see it very often, can define the synonym Gpot by writing

```
#define Gpot (GetPotG()/4)
```
// no ; at the end, it is not an instruction

What if the mountaineer passes through the mountain? We can decide that it is missed and we start again, we can subtract the value, we can not subtract but add negative points, we can ignore. The simplest is to add up the absolute value of the deviations. There is a function C for this:

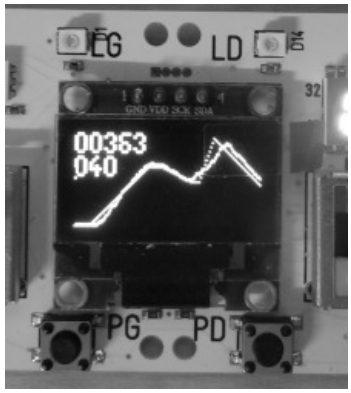
`abs(a,b);` calculate **`a-b`** if **`a>=b`** and **`b-a`** if **`a<b`**.

The program starts by drawing the mountain, but the screen can not be read to find out where the mountain is, and it would be complicated. The simplest is to redraw, while displaying
a Dot () according to the potentiometer and performing the calculation of the error. So for the first trip:

```
Repeat (10) {DDot (x++,63-y); Dot(x,63-Gpot; tot += abs(Gpot-y); Del;}
```
flat, y does not change — according to pot — error calculation — delay and display

Del; is also a synonym, a definition that will allow us to change this time in one place at the beginning, and the update will be done in all lines. With a delay of 100ms, we follow quite easily.
In the definition of Del, we add the display of the score. We have also decided that the delay is a variable called dl, initialized at 50ms, and decreasing by 5 at each part. A down counter also prepares the start of the trek. It uses the Digit (n) function; whose parameter is the displayed decimal or hexadecimal digit. The game starts again by accelerating the movement, thus reducing the delay from 50 to 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;
   Repeat (10) {Dot (x++,63-0);}        // plateau
   Repeat (40) {Dot(x++,63-y++);}       // 45 degrés
   Repeat (10) {Dot(x++,63-y);}         // plateau
   Repeat (20) {Dot(x++,63-y); if(x%2){y--;}  }  // petite descente
   Repeat (5)  {Dot(x++,63-y);}         // plateau
   Repeat (10) {Dot(x++,63-y); y+=3;}   // montée raide
   Repeat (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;
   Repeat (10) {DDot(x,63-Gpot); Dot (x++,63-y);  tot += abs((Gpot)-y); Del;}
   Repeat (40) {DDot(x,63-Gpot); Dot(x++,63-y++); tot += abs((Gpot)-y); Del;}
   Repeat (10) {DDot(x,63-Gpot); Dot(x++,63-y);   tot += abs((Gpot)-y); Del;}
   Repeat (20) {DDot(x,63-Gpot); Dot(x++,63-y); if(x%2){y--;} tot += abs((Gpot)-y); Del;}
   Repeat (5)  {DDot(x,63-Gpot); Dot(x++,63-y);   tot += abs((Gpot)-y); Del;}
   Repeat (10) {DDot(x,63-Gpot); Dot(x++,63-y); y+=3; tot += abs((Gpot)-y); Del;}
   Repeat (33) {DDot(x,63-Gpot); Dot(x++,63-y--); tot += abs((Gpot)-y); Del;}
   dl -=5; if (dl==0) dl=50;   // accélère
   Attend(PousG);
}
```



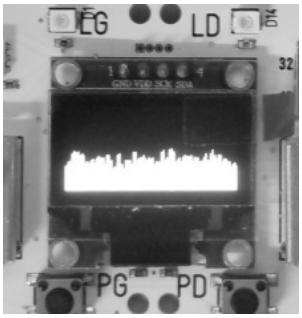
Note: Si vous connaisez le C, `Repeat(nn){}` est la sténo pour `for(byte i=0, i<nn; i++) {}`

Random numbers

Arduino offers the function random (Min, Max); which makes a number between Min and Max-1.
This number is calculated, it follows the same sequence if we do not change an initial parameter with the function randomSeed (value). For more details, see
https://en.wikipedia.org/wiki/Random_seed and the simple examples in www.didel.com/EduMod3.pdf.

Let's draw numbers randomly, from 0 to 127. Let's count how many times each number appears. And show the result on the Oled as a histogram. Let us construct in the position x = 13 a vertical segment whose length is equal to the number of times the 13 was fired. We need 127 counters, grouped in a table defined by: byte taHisto [128]; At each draw, mark the corresponding point on the screen.

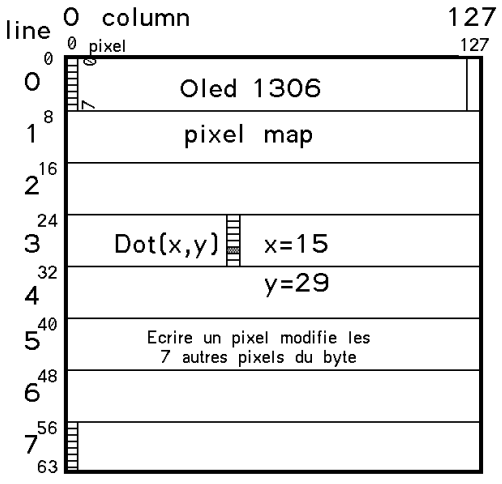
```
//FunOledRandom.ino  essai vecteur horiz
#include "Fun.h"
#include "Oled.h"
void setup () { SetupFun(); SetupOled(); randomSeed(GetPotD()); }

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);
}
```



Display numbers and texts

If you know the "terminal" Arduino, the Oled gives the same facilities, with the advantage of a fixed screen, but obviously we must say each time on which line and from which column we write.
Large characters (Big ..) take 2 lines high and do not go to line 0.
LiCol (li, col) with li 0-7 and 0-127 position the pointer.
a The functions are

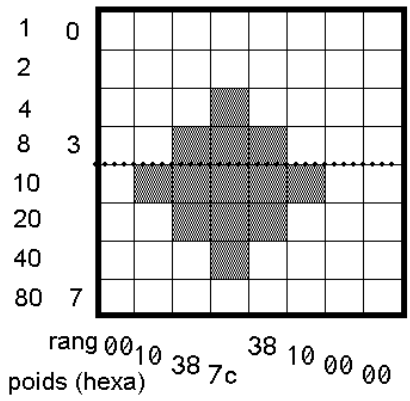
```
Car('a');  Text("txt");  Sprite(smile);
Bin8(v8); Hex8(v8); Hex16(v16);
Dec8(v8); Dec16(v16); Dec9999(v13);
Big('a');  BigBin8(); BigHex8(); BigHex16();
BigDec8();  BigDec9999();

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



Sprites

A sprite must be defined in a table. This table is in the library for sprites Smile, and Sad, called by Sprite (Smile);
If the sprite is in the user's program memory, like the cross opposite, you have to declare
byte cross [] = {0x00,0x10,0x38,0x7c, 0x38,0x10,0x00,0x00};
The instruction to copy in the center of the screen is
Halter (3.60); MySprite [cross];
The length is obviously less than 127.

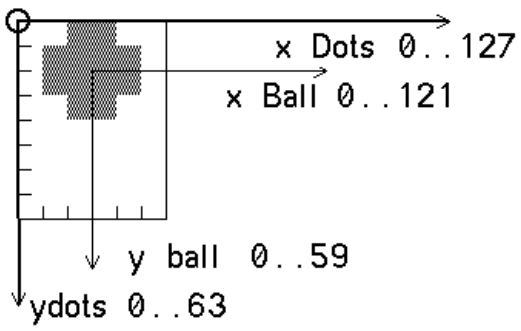
For a sprite of 16 high, it is necessary to declare 2 sprites of 8 of height and to align them with two LiCol (,) ;.
You can not easily move a sprite. To erase it, you have to define an empty goblin of the same size.



4x4 ball

The Ball (x, y) function is an object of 4x4 pixels in a field of 6x6 so that moving a box in all directions erases the previous ball. However, if the move is fast, the app may require a move of more than one unit, which will leave a poop.

The reference point is in the center of the ball, so it can be moved along 122x60 positions.
The function Ball (x, y); positions the ball on the field and saturates at 121/59. It uses global variables x, y (declared in Oled.h or Ping.h).



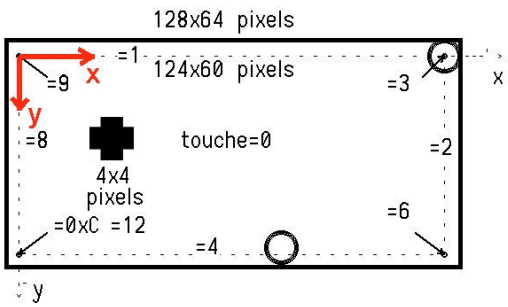
The functions to move the ball are delicate, and there are inevitably cases of recovery. GFX libraries avoid these overlays, but are not fast enough to play ping pong.

Displacement test

The ball is moved with potentiometers G (horizontal) and D (vertical). The values are not saturated; easy to add if (x> 122) {y = 122;}, it will be necessary.
A frame is built with a bullet in each corner. We can check how the frame is demolished if we do not saturate. We also see that we have to go slowly, so that the ball moves 1 at a time.

```
//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
  }
}
```

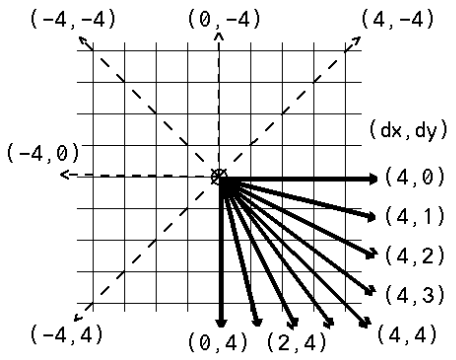
A very interesting function of the library is tested by this program. The Key () function; updates the global variable touch which is zero in the center of the screen, and takes the values indicated on the drawing when the ball touches an edge or a corner.



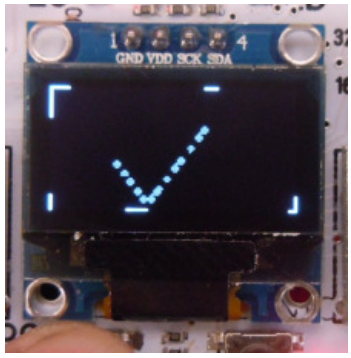
Playing with balls

You can multiply the balls on the screen, if they do not overlap. We can move them from +/- 1 at a time in x and y, Imagine games that fly Ball (x, y); and Dot (x, y); .

The function PosDir (x, y, dx, dy); defines the initial position of the ball and its direction, with for dx dy values of
-4 to +4, so only a few possible inclinations.
The Step () function; moves the ball one notch according to dx / dy and updates the global predefined touch variable which takes one of the 10 values according to the affected edge.
The speed of the ball depends on the time between two Step ();
This duration is the sum of the actions to be taken: the drawing of the ball is quite long (2ms) and it is completed by a delay during the development.



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:

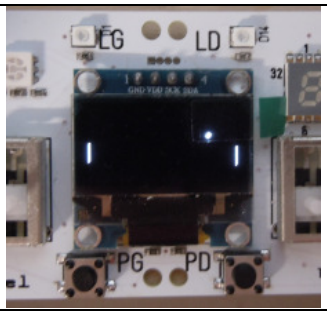



```
//FunBall.ino  2320 b 137v
#include "Fun.h"
#include "Oled.h"
void setup () { SetupFun(); SetupOled(); }

 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) ;
    }
  }
}
```

The binary coding used for touch is clever. If we are in the 9 = 8 + 1 corner, for example, two if conditions are satisfied and the ball goes back on itself, which is what we want. There is no need to predict these 4 cases.

Racket

For horizontal or vertical segments, Dot (x, y) is used in a Repeat loop (nn); (or for loop).
Snowshoes have a difficult constraint: writing a vertical line takes time and the option that minimizes this time leaves some noises if you move very quickly.
For ping-pong two rackets are programmed in x1 = 0 and x2 = 127.
The variable touch is updated in this case, and it is necessary to use the global variables y1 and y2 to position the 2 rackets.
y1 and y2 depend on PotG and PotD potentiometers.



The racket is 16 pixels high and its high point moves from 0 to 48. The formula (3x (256-Pot) / 16 gives a value from 0 to 48 with the value 0 when the pot is at the top.
The drawing of the racket lasts 3ms. The drawing of the ball is ~ 2ms. The minimum time to cross the screen is 127x8ms ~ = 1 second.

If the ball touches the rackets thus defined, the variable touch sees the bits 4 and 5 (value 0x10 = 16 and 0x20 = 32) activated. We have everything we need now to program 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;
}
```

We see that the "speed of displacement pots" v1 and v2 is calculated. The value seems to be between 0 and 5. The idea is to cut the bullets, and act on the return angle and / or the speed.
More simple to program with what we have seen, we can wait at startup on a pusher and change the speed or angle of departure. Improve the texts, display a score is easy. But while the ball moves, the slightest display will slow down.

Other games?

There are simple reflex programs, like programs 7 and 10 of the demo.
You can play with light shows, random numbers, program Simon's game and there are a lot of ideas on the web.

For a breakout and other programs on screen. there are either superposition or speed constraints.
Developing primitives like those needed for our ping-pong requires a lot of thinking and optimizations.

Note concerning the library "Fun.h"

The declared functions also contain functions specific to the Edu-C card (EduC.h library), functions that replace C instructions and simplify the understanding of the programs.
The Oled.h graphic library has evolved during development and is not stabilized. Each program contains the libraries it uses; they do not have to be identical.

```
Repeat(nn) {instructions};  same as  for(byte i=0;i<nn;i++}
LoopIf(){ }  identique à  while(condition true) {instructions}
Stop();  identique to while(1); WaitTill(cond) same as à while(!cond);
WaitTillPousG();  AttPousD();  AttPousG(){faire en attendant}   Attente pression
```

What to know if you have never programmed in C

Lowercase and uppercase letters are important.
The instructions end with one semicolon **;**
// announce a comment until the end of the lineles variables sont déclarées comme byte ou int.

Constant and variable parameters are in parentheses. The parenthesis is empty if there is none.
The true / false conditions are in parentheses (true?)
The groups of instructions are in braces {do}
You can put several instructions per line, if that makes it easier to read.
We must shift and align the groups of instructions.
For ease of calculation, v ++ takes the value v and adds one for later.
The first two lines will always be the same (C / Arduino constraints).
Similarly, the instructions are in Arduino's loop structure; loop () {instructions}
Variables and functions must be defined before being used.

The program FunLum.ino in the FunLum folder makes three passes on the Leds in all or nothing, then on these same leds in proportional, also using the Repeat function, derived from the for loop of C.
At the end of the program, we wait for a pressure on the PousG to start again.
Three times in milliseconds are declared, they can be changed to change the pace.

```
//FunLum.ino Light 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();
  Repeat(3) {LedGOn; Dn; LedDOn; Dn; LedGOff; Dn; LedDOff; Dn;}
  byte v5=0;
  Repeat(32){LedG(v5++);Dc;}  Repeat(32){LedG(--v5);Dc;}
  Repeat(32){LedD(v5++);Dc;}  Repeat(32){LedD(--v5);Dc;}
  Repeat(32){Rouge(v5++);Dc;} Repeat(32){Rouge (--v5);Dc;}
  Repeat(32){Vert(v5++);Dc;}  Repeat(32){Vert(--v5);Dc;}
  Repeat(32){Bleu(v5++);Dc;}  Repeat(32){Bleu(--v5);Dc;}
  AttPousG);
}
```

Mix table - Programme FunMix

Both knobs give an 8-bit value (from 0 to 255). We can use to mix the colors. As there are only two potentiometers, we will use the 2 pushers to choose on which color to act

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

To choose the pusher, use the if statement: if (PousG) {do what you need if in a hurry}.
GetPotG () and GetPotD () give the 8-bit value of the knobs. To have a value from 0 to 31, divide by 8 (8 * 32 = 256).

```
//FunMix.ino Color mix
#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);
}


}
```