## 4) Tira de LEDs direccionables

### 4.1) LEDs i NeoPíxels

Un LED és un diode, una unió PN de silici, amb 2 pins, un ànode i un càtode, que al donar-li corrent correctament emet llum. N'hi ha de diversos colors, tamanys i formes.
Tenim també els diodes multicolor, que realment són tres LEDs junts, un vermell, un verd i un blau, que permeten representar qualsevol color RGB variant la intensitat de llum de cadascun dels LEDs. Tenen 4 pins, un ànode per cada color i un càtode comú.
Si penséssim en muntar una tira de LEDs multicolor a on no tinguessin tots un mateix color sinó que els volguéssim controlar independentment, necessitariem 3 cables per cada LED més 1 pel comú. Si imaginem que creem una tira amb 10 LEDs, necessitarem posar 31 cables i tenir un dispositiu que ho controli, tipus Arduino, amb 30 pins GPIO. Oi que sona massa enrevessat?
Per això es creen els LEDs RGB direccionables individualment, a on Adafruit ha popularitzat els seus models anomenats NeoPíxels. En general, són LEDs que incorporen un circuit integrat a dins seu que fa possible que siguin controlats amb un bus d'un sol pin, per a on transmeten en sèrie la informació d'un LED cap al següent. El controlador que porta cada LED integrat és el que determina el tipus de tira. Per a 5V utilitzarem el WS2812.

### 4.2) Arrays

Un array no el definiria com pròpiament un tipus de variable, sinó com un complement a un tipus de variable ja existent. Defineix un grup d'elements iguals que són accedits per indexació.
Podem tenir variables sueltes tipus *char*, *integer*, *float*, *boolean*, però si ens en calen moltes del mateix tipus per a mantenir un llistat, haurem de crear un array per a accedir-hi de forma simple i eficient. Per exemple, crearem *int notes[40]* per a definir un llistat de valors numèrics per a les notes d'una classe amb 40 alumnes. Amb un sol nom de variable realment n'hem definit 40, i ens permetrà programar algoritmes per trobar la nota més alta o la més baixa, per exemple, o sigui, poder operar les dades fàcilment i no tenir 40 variables amb noms diferents que faria complicadíssim, per no dir impossible, el codi per a tractar-les.
Aquí va un exemple per a calcular de forma molt curta i senzilla la mitjana de les notes dels alumnes, recorrent d'extrem a extrem l'array utilitzant un bucle *for* :

```
int notes[40];
int sumatori, mitjana;
int i;

sumatori = 0;
for (i=0; i<40; i++) {
   sumatori = sumatori + notes[i];
}
mitjana = sumatori / 40;
```

Observar que els arrays s'indexen des de zero, en l'exemple de 0 a 39, o sigui 40 valors.
Podem definir arrays múltiples. Els més habituals són els de dues dimensions, *int valors[4][4]*, per a tractar càlculs matemàtics de matrius, i de tres dimensions, *int grafic[10][10][10]*, al gestionar dibuixos en 3D.
Un dels tipus d'arrays més utilitzats són els *arrays de chars*, una llista de caracters quasi bé equivalent al tipus de variables *string*. La diferència és que els *strings* es defineixen sense longitud i dinàmicament van agafant més o menys memòria en funció de la paraula o frase que s'hagi d'emmagatzemar en cada moment. En canvi, els *arrays de chars* tenen la longitud fixada quan els definim i no es pot superar o el programa deixarà de treballar correctament. Utilitzarem *strings* quan necessitem flexibilitat amb el seu contingut, i *arrays de chars* quan calgui indexació. Cal saber que els *strings* també poden ser llegits i escrits per índex, però com que la seva longitud és variable en el temps, s'ha de saber ben bé la longitud que té en tot moment, ja que tampoc pot ser superada.

```
char paraula[15];
String frase;

frase = "Hola. Bon dia ";
paraula[0] = 'M';
paraula[1] = 'a';
paraula[2] = 'k';
paraula[3] = 'e';
paraula[4] = 'r';
paraula[5] = 115;     // codi numèric ASCII equivalent a la 's'
paraula[6] = 0;       // valor de final de cadena
frase = frase + paraula;    // frase serà "Hola. Bon dia Makers"
frase[0] = 'A';
frase[1] = 'd';
frase[2] = 'e';
frase[3] = 'u';     // frase contindrà "Adeu. Bon dia Makers"
```

A la tira de LEDs farem servir un array per a especificar posició a posició de quin color caldrà pintar cadascun dels LEDs. La longitud de l'array serà doncs el nombre total de LEDs que volguem controlar. El tipus de variable serà una nova, la RGB (Red Green Blue), inclosa a la llibreria que utilitzarem, que amb aquests tres valors ens permetrà seleccionar entre un ampli espectre de colors.

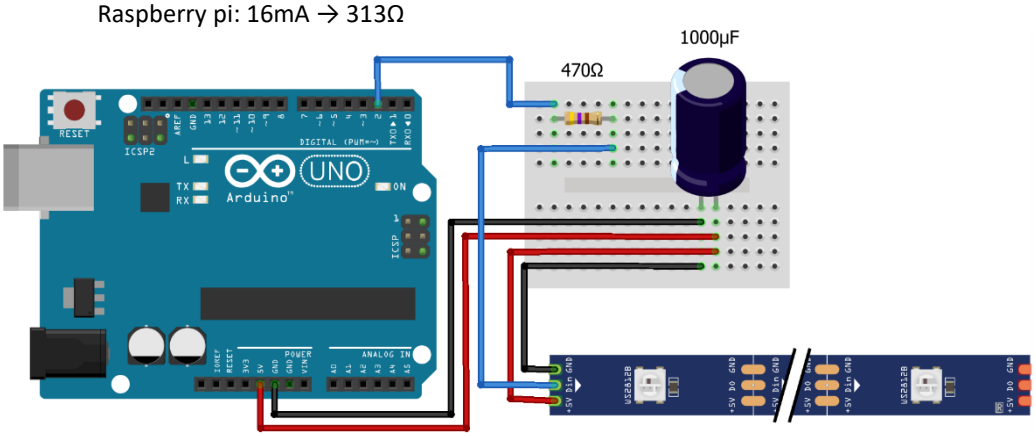
### 4.3) Connexió de la tira LED

El circuit bàsic necessari per a fer funcionar una tira de LEDs direccionable és molt senzill. Disposa de 3 cables, dos per alimentar-la, amb 5V les WS2812B que farem servir, i un per a controlar-la.

Observar que la tira té pintada una fletxa que indica la direcció d'indexació dels LEDs i per tant l'extrem des d'on li donarem les ordres de control. L'altre extrem quedarà lliure o bé se li podrà connectar una altra tira per augmentar la longitud, amb la precaució de subministrar més alimentació de 5V a cada tram de 5 metres.

És recomanable afegir un condensador de valor elevat entre els pins de l'alimentació, per ajudar a compensar els consums extrems que es poden produir a l'encendre i apagar els LEDs de cop i que ens podrien fer resetejar l'Arduino o inclús espatllar l'alimentació. Un condensador electrolític sobre els 1000µF serà una bona elecció. Compte al muntar-ho que aquests condensadors tenen polaritat. També protegirem el pin de control amb una resistència en sèrie que limiti el corrent, així en cas de curt-circuit no se'ns cremarà. Sabent els corrents màxims que suporten els pins GPIO i que el voltatge que poden rebre serà els 5V de l'alimentació dels LEDs, calcularem les resistències de protecció adeqüades:

Arduino: 40mA (20mA alguns models més senzills) → 125Ω (ó 250Ω)
ESP8266: 12mA → 417Ω
Raspberry pi: 16mA → 313Ω



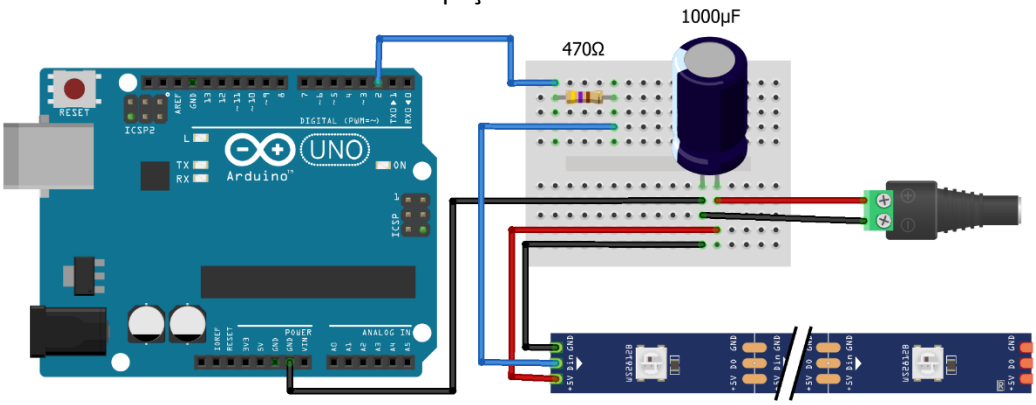

Cal mirar les especificacions de la tira que adquirim, i és aconsellable també fer unes mesures amb un amperímetre per alimentar-ho tot correctament i no sobrecarregar els alimentadors. El consum habitual de cada LED encès a màxima intensitat i de color blanc (situació extrema amb els tres RGB encesos), sol ser de 0,2W/LED. Com que van alimentats a 5V el consum màxim serà de 40mA/LED.

Amb l'Arduino alimentat mitjançant el port USB el màxim corrent que s'aconsella és de 500mA. Això només ens permetrà controlar una tira amb 12-13 LEDs. S'haurà de tenir molt en compte i implementar alguna proposta de solució per abarcar més LEDs:

- Controlar quants LEDs tenin engegats alhora, i no superar el límit.
- Baixar la lluminositat, encenent-los a un 20-40% ja sol ser suficient i fan prou llum.

  Hem realitzat les següents mesures:

  10 LEDs en Blanc al màxim (255) → 365mA
  10 LEDs en Blanc al 40% (100) → 153mA
  10 LEDs en Blanc al 20% (50) → 84mA
  30 LEDs en Blanc al 40% (100) → 430mA
  30 LEDs en Blanc al 20% (50) → 224mA
- Utilitzar una alimentació externa capaç de subministrar més corrent:

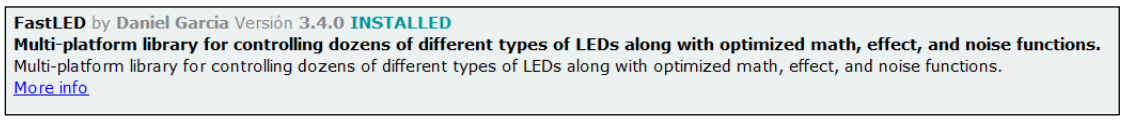



### 4.4) La llibreria FastLED

Utilitzar la llibreria FastLED, http://fastled.io/ , fa que el codi per a controlar tires de LEDs es torni molt simple, i ens proporciona eines adicionals per a poder fer operacions complexes més ràpid. És una llibreria molt potent però amb les següents instruccions ja podem fer projectes interessants.

Primer de tot cal importar-la a l'IDE d'Arduino:

Programa / Incluir Librería / Administrar Bibliotecas/ *buscar per fastled*




Dins el nostre programa caldrà realitzar 4 passos inicials:

- Incluore la llibreria.

```
#include "FastLED.h"
```

- Definir el nombre de LEDs de la tira i el pin des d'on la controlarem.

```
#define PIN_TIRA_LED 2
#define NUM_LEDS 10
```

- Crear un array RGB que defineixi la tira de LEDs amb la longitud anteriorment definida.

```
CRGB leds[NUM_LEDS];
```

- Inicialitzar la tira de LEDs associada a l'array amb el format propi de la llibreria.

```
FastLED.addLeds<NEOPIXEL, PIN_TIRA_LED> (leds, NUM_LEDS);
```

Després, aplicarem seguretat per evitar disgustos:

- Definim el Voltatge i el màxim Amperatge que permetem, 500mA de l'USB, en l'exemple.

```
FastLED.setMaxPowerInVoltsAndMilliamps(5, 500);
```

  En superar-se l'amperatge establert, es baixarà la lluminositat de tota la tira el necessari per seguir cumplint el màxim consum seleccionat.
- Establim l'escala global de lluminositat.

```
FastLED.setBrightness(100);
```

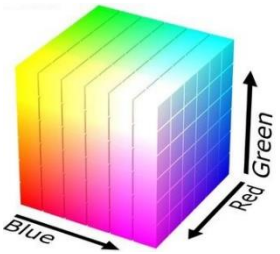
  Tots els LEDs s'encendràn en proporció a aquest límit. És a dir, si al LED li assignem el valor màxim 255 s'encendrà realment a 100, i si li assignem un 120 s'encendrà a 47.

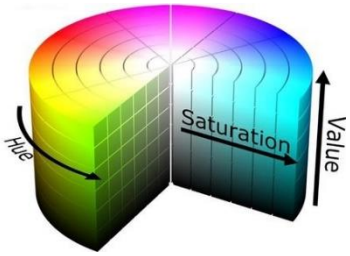
I ja per a controlar-ho tindrem unes comandes bàsiques que utilitzarem per a modificar els valors i poder així encendre un a un tots els LEDs i representar el que volguem:

- Establir un color al LED de la posició "i", per exemple posar-lo en verd. Tenim vàries formes d'assignar-ho, les més habituals són aquestes:

```
leds[i] = CRGB(0, 255, 0);
leds[i] = CHSV(96, 255, 255);
leds[i] = CRGB::Green;
```

* per RGB: Red, Green, Blue
* per HSV: Hue (Tonalitat), Saturation, Value
  ó també anomenat HSB: Brightness (Lluminositat)
* per color directe que té definits la pròpia llibreria




```
leds[i].red = 0;
leds[i].green = 255;
leds[i].blue = 0;
```

```
leds[i].r = 0;
leds[i].g = 255;
leds[i].b = 0;
```

* o un a un cada component del color, en qualsevol de les dues formes possibles

- Posar tots els LEDs de la tira d'un mateix color.

```
fill_solid(leds, NUM_LEDS, CRGB::Red);
```

Podem fer-ho manualment amb un bucle "*for*" assignant valors un a un amb alguna de les formes de la comanda explicada anteriorment, però la lliberia ja té aquesta instrucció per a fer-ho de forma directa. De fet, canviant NUM_LEDS per un valor, els pintarem des de l'inici fins aquella posició.

- Apagar tots els LEDs.

```
FastLED.clear();
```

En certes ocasions ens cal apagar tots els LEDs, principalment a l'inici per començar des de zero, o en algunes aplicacions en què és més senzill apagar tot i encendre el que volem i no pas haver de controlar el que cal esborrar respecte l'estat anterior. Podem fer un *fill_solid* de color *Black*, prou utilitzat en certs exemples, però tenim aquesta comanda que fa el mateix i el codi queda més clar i llegible.

- Actualitza i aplica els valors establerts a la tira.

```
FastLED.show();
```

Amb totes les comandes anirem modificant l'array amb el contingut que volem visualitzar, però fins que no executem aquest *show* no s'aplicaran els canvis. Això ens permetrà canviar de cop a la nova representació i fará que no es vegi com es va actualitzant, que crearia un efecte pampalluga lent i extrany.
Cal esmentar que el `FastLED.clear(true);` amb l'argument *true*, apaga tots els LEDs i ja aplica l'acció sense necessitar del *show*.

### 4.5) Aprenem a programar-la

Intentarem utilitzar una manera didàctica pas a pas amb reptes petits i assolibles per anar creant i millorant el codi fins a obtenir un resultat amb un efecte creatiu.

Primer de tot, tal i com hem explicat a l'apartat anterior, tindrem un bloc de codi més o menys fixe, a on inclourem la llibreria, definirem constants, la variable que representa la tira de LEDs, el control de la seguretat i apagarem tots els LEDs per a inicialitzar-ho.

```
#include "FastLED.h"

#define PIN_TIRA_LED 2
#define NUM_LEDS 30
#define LONG_CUA 5
#define ESPERA 100

CRGB leds[NUM_LEDS];

void setup() {
   FastLED.addLeds<NEOPIXEL, PIN_TIRA_LED> (leds, NUM_LEDS);
   FastLED.setMaxPowerInVoltsAndMilliamps(5, 500);
   FastLED.setBrightness(50);
   FastLED.clear(true);
}
```

A partir d'aquí jugarem a plantejar-nos què volem fer i fer-ho, tot aprenent equivocant-nos.

**a)** Encenem un LED, el de la posició 20 per exemple, de color blau:

```
void loop() {
   leds[20] = CRGB(0, 0, 255);
}
```

Ho provem i sembla que no fa res. Hem definit que volem un LED encès però no hem aplicat els canvis. Provem-ho de nou:

```
void loop() {
   leds[20] = CRGB(0, 0, 255);
   FastLED.show();
}
```

Ara ja s'encén, però si comptem és el de la posició 21, no la 20. Recordeu que comencem comptant des del 0 i no de l'1. No canviarem el codi però ha servit per a reforçar el concepte que el necesitarem per més endavant.

**b)** Encenem tots els LEDs de la tira un a un del mateix color, posant un petit retràs per a que es vegi com va avançant l'efecte.

```
void loop() {
   for(int posicio=0; posicio<NUM_LEDS; posicio++) {
      leds[posicio] = CRGB(0, 0, 255);
      delay(ESPERA);
   }
   FastLED.show();
}
```

Efectivament necessitarem un bucle for amb una variable posició que recorri d'inici a fi tot l'array i a cada iteració assignem el color i posem un retràs.
Ho provem i sí, ens ho ha encès tot, però no hem vist la progressió de com ho ha fet.
Potser caldrà posar el show a dins el bucle, per que a cada canvi individual ho mostri.

```
void loop() {
   for(int posicio=0; posicio<NUM_LEDS; posicio++) {
      leds[posicio] = CRGB(0, 0, 255);
      FastLED.show();
      delay(ESPERA);
   }
}
```

Genial, ja el veiem progressar i queda tot encès. No ho hem demanat a l'enunciat, però què tal si un cop arribat al final l'apaguem i tornem a repetir l'encesa?

```
void loop() {
   FastLED.clear();
   for(int posicio=0; posicio<NUM_LEDS; posicio++) {
      leds[posicio] = CRGB(0, 0, 255);
      FastLED.show();
      delay(ESPERA);
   }
}
```

Hi ha vàries maneres de fer-ho i segur que totes vàlides, posant per exemple l'apagat al final del codi que per lògica caldria apagar-ho un cop ho hem encès tot i sembla que hagi de ser l'última acció. O potser la més entenedora sigui esborrar-ho al començament per partir de zero a l'inici. En ambdós casos sense necessitat de fer un show ja que esborrant l'array ja n'hi haurà prou.

**c)** Fem un pas més i mantenim un sol LED encès que recorri tota la tira.

```
void loop() {
   for(int posicio=0; posicio<NUM_LEDS; posicio++) {
      FastLED.clear();
      leds[posicio] = CRGB(0, 0, 255);
      FastLED.show();
      delay(ESPERA);
   }
```

Correcte, canviem de lloc l'esborrat total de la tira i ho fem a cada iteració abans d'encendre el que ens interessa.

**d)** Va, fem ara un cuc i que siguin dos els LEDs que estan encesos alhora.

```
void loop() {
   for(int posicio=0; posicio<NUM_LEDS; posicio++) {
      FastLED.clear();
      leds[posicio] = CRGB(0, 0, 255);
      leds[posicio-1] = CRGB(0, 0, 255);
      FastLED.show();
      delay(ESPERA);
   }
```

D'acord, funciona, però amb tres puntualitzacions. 1) Quan arriba al final passa de 2 LEDs encesos a 0 de cop, i queda extrany. 2) Quan ens trobem a la posició 0 apaguem el de la posició -1, i això d'indexar un array fora de límits no es pot fer, potser per sort funciona però es corre un alt risc que el codi es pengi. Aquest segon punt s'ha d'arreglar sí o sí.

```
void loop() {
   for(int posicio=0; posicio<NUM_LEDS; posicio++) {
      FastLED.clear();
      leds[posicio] = CRGB(0, 0, 255);
      if ((posicio-1) >= 0) {
         leds[posicio-1] = CRGB(0, 0, 255);
      }
      FastLED.show();
      delay(ESPERA);
   }
```

El que farem és sols pintar aquest segon LED de la cua si comprovem que la seva posició sigui més gran o igual a zero, que és a l'extrem a on tenim el problema.
Molt bé, solucionat, però falta la tercera puntualització, que és molt important. 3) Què passa si volem una cua de longitud 5? Afegim més línies duplicant el codi? I si necessitem que a cada volta completa la cua sigui un punt més llarga? el codi no pot crèixer sol. Aquesta solució és limitada i poc flexible, haurem de buscar un altre mètode.

**e)** Fem un pas enrera i tornem a treballar amb un sol LED recorrent tota la tira. Pensem a no apagar-ho tot, sinó sols el que calgui controladament.

```
void loop() {
   for(int posicio=0; posicio<NUM_LEDS; posicio++) {
      leds[posicio] = CRGB(0, 0, 255);
      leds[posicio-1] = CRGB(0, 0, 0);
      FastLED.show();
      delay(ESPERA);
   }
}
```

Molt bé, apaguem la posició anterior i llavors sols l'actual queda encès. Però observeu que hi ha un petit defecte, que l'últim LED mai s'apaga, perquè s'apagaria quan encenguèssim el següent que no existeix i per tant mai hi accedim. I tornem a tenir el problema de la incorrecta indexació que ja sabem solucionar amb un if.

```
void loop() {
   for(int posicio=0; posicio<NUM_LEDS; posicio++) {
      leds[posicio] = CRGB(0, 0, 255);
      FastLED.show();
      delay(ESPERA);
      leds[posicio] = CRGB(0, 0, 0);
   }
}
```

Mireu què senzill que ha estat, sols hem canviat de lloc una línia de codi, deixant apagada aquesta posició de l'array que es mostrarà amb el show del següent cicle. I tots dos defectes han quedat solucionats.

**f)** Ara ja podem pensar de nou amb el cuc de 2 posicions enceses alhora.

```
void loop() {
   for(int posicio=0; posicio<NUM_LEDS; posicio++) {
      leds[posicio] = CRGB(0, 0, 255);
      FastLED.show();
      delay(ESPERA);
      leds[posicio-1] = CRGB(0, 0, 0);
   }
}
```

Molt enginyòs, distanciem la posició que apaguem i així en tenim més d'encesos. Què fàcil que ha estat, però tornem a tenir els dos primers problemes anteriors, LED final que no s'apaga i mala indexació. Provem de fer que el cuc sigui més llarg i veurem com es penja i queda bloquejat. És imperatiu afegir un if per arreglar la indexació a l'extrem inferior.

```
void loop() {
   for(int posicio=0; posicio<NUM_LEDS; posicio++) {
      leds[posicio] = CRGB(0, 0, 255);
      FastLED.show();
      delay(ESPERA);
      if ((posicio-LONG_CUA) >= 0) leds[posicio-LONG_CUA] = CRGB(0, 0, 0);
   }
}
```

I apaguem també els LEDs finals allargant la posició a la que arribem dins el for i vigilant llavors amb un altre if per no indexar malament a l'extrem superior.

```
void loop() {
   for(int posicio=0; posicio<NUM_LEDS+LONG_CUA; posicio++) {
      if (posicio < NUM_LEDS) leds[posicio] = CRGB(0, 0, 255);
      FastLED.show();
      delay(ESPERA);
      if ((posicio-LONG_CUA) >= 0) leds[posicio-LONG_CUA] = CRGB(0, 0, 0);
   }
}
```

**g)** L'efecte ja comença a ser bo. Per fer-lo ben cíclic caldria millorar que quan tot just marxa per l'extrem final ja comenci a aparèixer per l'inicial.

```
void loop() {
   for(int posicio=0; posicio<NUM_LEDS+LONG_CUA; posicio++) {
      if (posicio < NUM_LEDS) leds[posicio] = CRGB(0, 0, 255);
      else leds[posicio-NUM_LEDS] = CRGB(0, 0, 255);
      FastLED.show();
      delay(ESPERA);
      if ((posicio-LONG_CUA) >= 0) leds[posicio-LONG_CUA] = CRGB(0, 0, 0);
   }
}
```

Quan deixa de cumplir l'if, encenem posició-NUM_LEDS amb un else, que equival al principi de la tira. Ens apareix un nou entrebanc, i és que com que ja hem anant avançant posicions respecte l'inici, al tornar a començar el cicle ho dupliquem i sembla que el moviment quedi parat un moment.

```
void loop() {
   for(int posicio=LONG_CUA; posicio<NUM_LEDS+LONG_CUA; posicio++) {
      if (posicio < NUM_LEDS) leds[posicio] = CRGB(0, 0, 255);
      else leds[posicio-NUM_LEDS] = CRGB(0, 0, 255);
      FastLED.show();
      delay(ESPERA);
      if ((posicio-LONG_CUA) >= 0) leds[posicio-LONG_CUA] = CRGB(0, 0, 0);
   }
}
```

Començant el for des d'aquell punt i no de la posició 0 ho soluciona.
Podem optimitzar el codi si indexem amb el mòdul de la posició, és a dir, amb la resta de la divisió dels dos nombres, de manera que quan sobrepassem el límit tornem a començar i no indexem fora de marges, amb el que no ens caldrà controlar-ho amb l'if-else i simplifica el codi.

```
void loop() {
   for(int posicio=LONG_CUA; posicio<NUM_LEDS+LONG_CUA; posicio++) {
      leds[posicio % NUM_LEDS] = CRGB(0, 0, 255);
      FastLED.show();
      delay(ESPERA);
      if ((posicio-LONG_CUA) >= 0) leds[posicio-LONG_CUA] = CRGB(0, 0, 0);
   }
}
```

**h)** Ja hem aconseguit l'objectiu, i ens funciona prou bé. Per anar evolucionant poc a poc una idea ha estat un procés molt interessant, superant tots els esculls que hem trobat i hem après a resoldre. Però realment no és la solució final que un programador escolliria. Hi ha diferents raons per a justificar aquesta afirmació, però la més decisòria seria el fet de no acabar de controlar el què s'està pintant. El mètode de deixar el que està pintat i esborrar a distàcia allò antic fa no saber massa el que visualitzem en cada moment i sempre es prefereix esborrar-ho tot i pintar el cuc sencer de nou cada vegada. Caldrà llavors utilitzar un segon for per a recòrrer tota la cua i encendre un a un tots els LEDs del cuc.

```
void loop() {
   for(int cap=0; cap<NUM_LEDS; cap++) {
      FastLED.clear();
      for(int cua=0; cua<LONG_CUA; cua++) {
         if ((cap-cua) >= 0) leds[cap-cua] = CRGB(0, 0, 255);
      }
      FastLED.show();
      delay(ESPERA);
   }
}
```

Aquí ja hem solucionat la correcta indexació amb un if per a que no se'ns bloquegi. Ara hem d'arreglar 1) que desaparegui pel final, 2) que tot just se'n va ja aparegui per l'inici, i 3) que aquest inici sigui a la posició correcta. Caldrà aplicar vàries de les solucions anteriors per a deixar-lo ben acabat.

```
void loop() {
   for(int cap=LONG_CUA; cap<NUM_LEDS+LONG_CUA; cap++) {
      FastLED.clear();
      for(int cua=0; cua<LONG_CUA; cua++) {
         leds[(cap-cua)%NUM_LEDS] = CRGB(0, 0, 255);
      }
      FastLED.show();
      delay(ESPERA);
   }
}
```

I com que cap-cua mai serà negatiu, ja que comencem el cap a una posició avançada, no ens caldrà patir per la incorrecta indexació a l'inici.

**i)** Més idees per a seguir aprenent:

- Feu que el cuc vagi rebotant de costat a costat.
- Manteniu la trajectòria cíclica però poseu 2 cucs separats i de diferent color.
- Investigueu la comanda millis() i feu que cadascun dels 2 cucs tingui una velocitat diferent.

## 4.6) Projectes

### 4.6.1) Cuc de Llum

Comencem incloent la llibreria, definint els valors constants i creant l'array que defineix la tira de LEDs.

```
#include "FastLED.h"

#define PIN_TIRA_LED 2
#define NUM_LEDS 30
#define ESPERA 50   // Regula la velocitat del moviment

CRGB leds[NUM_LEDS];
```

Dins el *Setup*, inicialitzem la tira, incorporem el màxim de corrent que permetem per seguretat, establim la lluminositat que desitgem i l'apaguem tota per partir de zero.

```
void setup() {
   FastLED.addLeds<NEOPIXEL, PIN_TIRA_LED> (leds, NUM_LEDS);
   FastLED.setMaxPowerInVoltsAndMilliamps(5, 500);
   FastLED.setBrightness(50);
   FastLED.clear(true);
}
```

Ja dins el *Loop*, creem dos bucles. Un per fer l'anada, que va del primer LED (a la posició zero) fins al darrer (posició NUM_LEDS-1 → <NUM_LEDS). I l'altre per la tornada, des de l'últim (NUM_LEDS-1) fins a l'inicial (zero → >=0).
A cada iteració, esborrarem el contingut de la tira, encendrem el LED que toqui controlat per l'índex "i", aplicarem els nous valors i posarem un retràs per a tenir un moviment suau.

```
void loop() {
   for(int i=0; i<NUM_LEDS; i++) {
      FastLED.clear();
      leds[i] = CRGB(0, 0, 255);
      FastLED.show();
      delay(ESPERA);
   }
   for(int i=NUM_LEDS-1; i>=0; i--) {
      FastLED.clear();
      leds[i] = CRGB(0, 255, 0);
      FastLED.show();
      delay(ESPERA);
   }
}
```

Li hem establit el color blau RGB(0, 0, 255) per anar i el verd RGB(0, 255, 0) per tornar. Si volem anar canviant el color recorrent tot l'espectre cromàtic, ens serà més senzill utilitzar la nomenclatura HSV, en la qual sols modificant el Hue ja ens anirà variant el color. El farem anar de 0-255 multiplicant la posició "i" amb un salt adeqüat al nombre de LEDs de la tira, NUM_LEDS. Canviariem l'assignació del color per la següent:

```
leds[i] = CHSV(i*255/NUM_LEDS, 255, 255);
```

Ara li dibuixarem una cua que representi l'estela que va deixant al moure's. Definirem un valor amb la llargària, per poder-la modificar, que inclourem a l'inici del codi.

```
#define CUA_LEDS 5
```

I farem uns petits canvis dins el *Loop*:
1) Dins de cada bucle "i" d'anada i tornada, haurem d'afegir un altre bucle "j" per que a cada posició que anem avançant no sols en pinti un LED sinó tot el cuc de llargària CUA_LEDS.
2) Com que la cua sempre ens ha de quedar enrera, a l'anada la pintarem amb posicions [i-j] i a la tornada amb [i+j].
3) Tal i com hem comentat a l'apartat Arrays, hem de vigilar no indexar mai fora de l'array, i als extrems és a on hem de tenir-ho en compte. Per això, nosaltres recorrerem tot l'abast que ens calgui, però posarem un condicional que sols pinti el LED quan estem dins de la indexació correcta (>=0 i <NUM_LEDS).

4) El bucles principals "i" els haurem de fer més llargs al final per permetre que tota la cua acabi de marxar de la tira. Els allargarem una longitud CUA_LEDS positiva al pujar per sortir per sobre i negativa al baixar per sortir per l'inici.

```
void loop() {
   for(int i=0; i<NUM_LEDS+CUA_LEDS; i++) {
      FastLED.clear();
      for(int j=0; j<CUA_LEDS; j++) {
         if ( ((i-j) >=0) && ((i-j) < NUM_LEDS) ) {
            leds[i-j] = CHSV(i*255/NUM_LEDS, 255, 255);
         }
      }
      FastLED.show();
      delay(ESPERA);
   }
   for(int i=NUM_LEDS-1; i>=0-CUA_LEDS; i--) {
      FastLED.clear();
      for(int j=0; j<CUA_LEDS; j++) {
         if ( ((i+j) >=0) && ((i+j) < NUM_LEDS) ){
            leds[i+j] = CHSV(i*255/NUM_LEDS, 255, 255);
         }
      }
      FastLED.show();
      delay(ESPERA);
   }
}
```

L'últim detall que ens queda per fer és que la cua vagi reduïnt la lluminositat, per fer un efecte més creïble de moviment. Modificarem el paràmetre Valor del HSV amb un càlcul similar per salts al que hem fet pel canvi de color, fent ara que com més lluny estem del cap, més gran serà "j", i per tant més restem i més petit serà el Valor o Llum.

```
CHSV(i*255/NUM_LEDS, 255, (CUA_LEDS-j)*255/CUA_LEDS);
```

### 4.6.2) VU-Meter

Un VU-meter és una barra de llum, habitualment en vertical, que indica el nivell de volum d'una font d'àudio, de color verd a la part baixa per indicar que el so és baix i transformant-se en groc a mesura que va pujant fins arribar al límit superior en vermell.

Tindrem un inici de programa similar a l'anterior, amb la inclusió de la llibreria i les definicions de constants, a on n'afegim una de nova per definir el color verd com a inici. Utilitzarem 3 variables que defineixin el *Volum* de so mesurat, l'*Altura* proporcional de LEDs que encendrem i el *Color* que tindrà cadascun a mesura que anem pujant.

```
#include "FastLED.h"

#define PIN_TIRA_LED 2
#define NUM_LEDS 30
#define COLOR_INICI 100    // Verd
#define ESPERA 4

CRGB leds[NUM_LEDS];
int volum;
int altura;
int color;
```

El *Setup* serà idèntic: inicialització, protecció i esborrat inicial.

```
void setup() {
   FastLED.addLeds<NEOPIXEL, PIN_TIRA_LED> (leds, NUM_LEDS);
   FastLED.setMaxPowerInVoltsAndMilliamps(5, 500);
   FastLED.setBrightness(50);
   FastLED.clear(true);
}
```

I el *Loop* tindrà dues parts, una primera a on generem un nombre aleatori de 0 a 255 per a simular un nivell de volum de so. Li canviarem el rang per un de més petit de 0 a NUM_LEDS per encabir-lo en el nombre de LEDs de la nostra tira de manera que:

Volum = 0 → Altura = 0

Volum = 200 i superiors (fins a 255) → Altura = NUM_LEDS

Mapejant fins a 200 i no fins el valor màxim de 255 simulem un efecte de saturació de la barra al seu valor màxim quan teòricament ja hi ha distorsió del so.

Per precaució, afegim uns condicionals per assegurar que *Altura* mai sobrepassarà la indexació de l'Array, ni per sota ni per sobre.

```
void loop() {
   volum = random(0, 255);
   altura = map(volum, 0, 200, 0, NUM_LEDS);
   if (altura < 0) altura = 0;
   if (altura > NUM_LEDS) altura = NUM_LEDS;
```

A la segona part, esborrem el contingut de la tira, encenem el nombre de LEDs que indiqui la variable *Altura*, que és proporcional al *Volum* generat, i cadascun amb el *Color* adequat calculat per a cada posició, i ho apliquem.

```
   FastLED.clear();
   for(int i=0; i<altura; i++) {
      color = COLOR_INICI - i*COLOR_INICI/NUM_LEDS;
      leds[i] = CHSV(color, 255, 255);
   }
   FastLED.show();
   delay(ESPERA*NUM_LEDS);
}
```

Per a mostrar el color de la barra ens aprofitarem de nou de la senzillesa de la representació HSV, a on un valor de Hue sobre 100 representa el color verd i 0 el vermell, trobant els grocs entremig. Utilitzant aquests valors a la inversa farem que:

Altura = 0 → Hue = 100 (verd)

Altura = NUM_LEDS → Hue = 0 (vermell)

Respecte un COLOR_INICI verd=100, li anirem restant uns salts proporcionals al nombre de LEDs de la tira a mesura que anem pujant, fins arribar a dalt de tot amb el vermell=0.

Si ens hi fixem, a cada canvi de valor, com que aleatòriament poden ser molt diferents i bastant distanciats, es produeixen uns salts abruptes i poc reals, amb apagats i encesos massa de cop. Ho suavitzarem anant d'un valor al següent pujant o baixat un a un cada LED. Per això caldrà recordar quina era l'*Altura* prèvia. Canviarem l'anterior variable per aquestes dues noves:

```
int altura_old = 0;
int altura_new;
```

I el *Loop* tindrà igualment dues parts, la primera idèntica de generació de nombres aleatoris amb el corresponent mapejat, utilitzant el nou nom de la variable.

```
void loop() {
   volum = random(0, 255);
   altura_new = map(volum, 0, 200, 0, NUM_LEDS);
   if (altura_new < 0) altura_new = 0;
   if (altura_new > NUM_LEDS) altura_new = NUM_LEDS;
```

I la segona part de pintat dels LEDs, però fent-ho diferent, abans ho esborràvem tot i pintavem de nou des de baix fins l'altura que tocava, i ara, si cal seguir pujant perquè la nova altura és superior a l'anterior, partirem des d'aquell punt i pintarem sols els LEDs que faltin per encendre, amb el color adeqüat.

```
    if (altura_new > altura_old) {
       for(int i=altura_old; i<altura_new; i++) {
          color = COLOR_INICI - i*COLOR_INICI/NUM_LEDS;
          leds[i] = CHSV(color, 255, 255);
          FastLED.show();
          delay(ESPERA);
       }
    }
```

Posant un retràs entre encesa de LED i LED per a que es vegi com puja i no ho fagi de cop, que deixariem perdre una mica de l'efecte que voliem millorar.

I si cal baixar perquè la nova altura és inferior a l'anterior, partirem des d'aquell punt i apagarem sols els LEDs que calgui, posant-los de color *Black*.

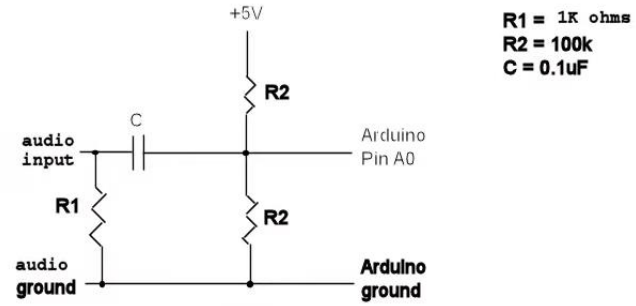
```
    else {
       for(int i=altura_old-1; i>=altura_new; i--) {
          leds[i] = CRGB::Black;
          FastLED.show();
          delay(ESPERA);
       }
    }
```

Finalment, actualitzem la variable *Altura anterior* per a preparar-nos pel següent cicle, i posem un retràs general que fagi que la durada de cada cicle sigui sempre el mateix independentment de si el salts són petits o grans i hem fet menys o més petits retrassos a dins de cada encesa o apagada.

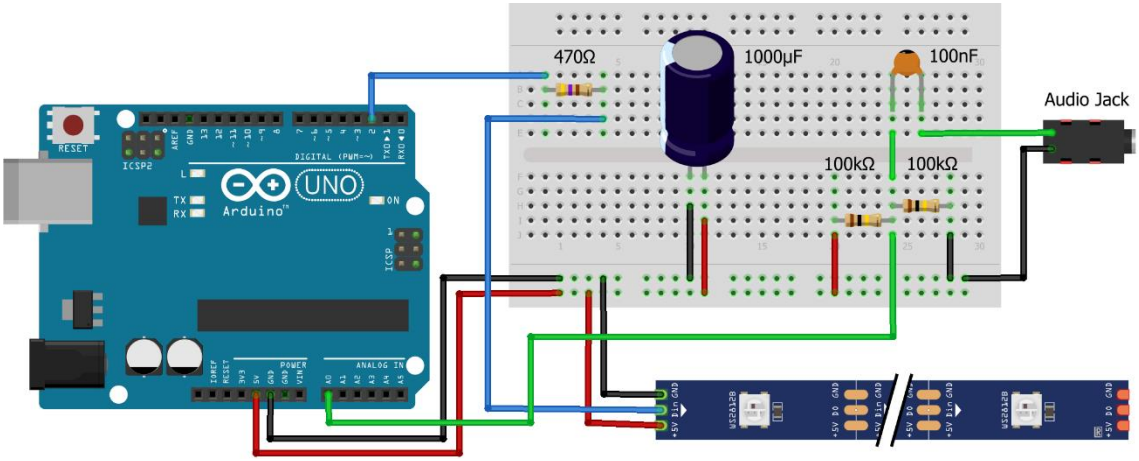
```
    altura_old = altura_new;
    delay(ESPERA*(NUM_LEDS - abs(altura_new - altura_old)));
}
```

El VU-meter ja ens funciona, però l'estem generant amb nombres aleatoris. Utilitzem el següent projecte de *PensActius.cat* per connectar-lo realment a una font de música i veure el resultat real. https://www.youtube.com/watch?v=6FUaNlQMFo0

Caldrà aplicar una certa electrònica de protecció per poder endollar un senyal d'audio a una entrada analògica de l'Arduino. Muntarem el següent circuit:




A on R1 no serà necessària, i ens quedarà el següent prototipat:



I al codi, senzillament definirem dins al *Setup* l'entrada analògica A0 que utilitzarem per a llegir l'audio,

```
pinMode(A0, INPUT);
Serial.begin(9600);
```

i dins el *Loop* farem la lectura real del pin analògic enlloc del *random* anterior.

```
volum = analogRead(A0);
Serial.println(volum);
```

Hem també inicialitzat el port sèrie i imprès els valors obtinguts, perquè tal i com explica molt bé el video del projecte al punt d' 1h i 12min utilitzant l'opció del *Serial Plotter* de l'IDE d'Arduino, caldrà "calibrar" el nostre circuit. Veurem quin valor llegim quan no tenim cap so (centre del senyal) i quin altre de valor màxim quan estem reproduint un àudio. Això ens donarà els marges reals del nostre so.
En les nostres proves, estem centrats sobre els 514 i arribem a pics de 590. Aquests seran els valors que mapejarem per calcular els LEDs que cal encendre. Ara, tot valor inferior a 514 generarà una *Altura* correcte però de valor negatiu, que haurem de convertir a positiva per no perdre la meitat de les dades del so rebut.
La primera part del *Loop* finalment ens quedarà així.

```
void loop() {
   volum = analogRead(A0);   // centrat a 514 i fins 590
   Serial.println(volum);
   altura_new = map(volum, 514, 590, 0, NUM_LEDS);
   if (altura_new < 0) altura_new = -altura_new;
   if (altura_new > NUM_LEDS) altura_new = NUM_LEDS;
```

Si utilitzem un PC amb Windows per a reproduir una cançó i endollem el cable d'àudio que ve del nostre vu-meter a la sortida de headphones, el so deixarà de sonar pels altaveus de l'ordinador, veurem com els LEDs segueixen el ritme de la música però no l'escoltarem. Caldrà utilitzar un altaveu bluetooth extern, sincronitzar-lo amb el PC i seguir aquest tutorial per a configurar Windows per a treure el so per dues sortides al mateix temps:

https://www.profesionalreview.com/2018/12/05/dos-salidas-de-audio-windows-10/

Ara que ja ho tenim funcionant, podem acabar-ho d'afinar. Si observem la gràfica del *Serial Plotter* veurem que el so que capturem, malgrat hem agafat la part inferior de la gràfica i l'hem posat a dalt, encara seguim tenint molta fluctuació que genera pujades i baixades massa exagerades. Estem mostrejant el senyal a massa freqüència i necessitem ponderar-ho. El que farem és agafar 50 lectures, per exemple, i fer la mitjana. Al tenir valors baixos ens caurà el nivell de senyal i ja no tindrem el màxim a 590 sino que ara el mesurem sobre 560. Canviarem doncs el valor del mapeig també.
Ens caldrà una variable nova a l'inici:

```
long int volum_acumulat;
```

I la primera part del *Loop* que definitivament ara serà:

```
void loop() {
   volum_acumulat = 0;
   for(int a=0; a<50; a++) {
      volum = analogRead(A0);
      // convertim el senyal "negatiu" a "positiu"
      if (volum < 514) volum = 514 + (514 - volum);
      volum_acumulat += volum;
      delay(1);
   }
   volum = volum_acumulat/50;   // Calculem la mitjana
   Serial.println(volum);
   altura_new = map(volum, 514, 560, 0, NUM_LEDS);
   if (altura_new < 0) altura_new = -altura_new;
   if (altura_new > NUM_LEDS) altura_new = NUM_LEDS;
```

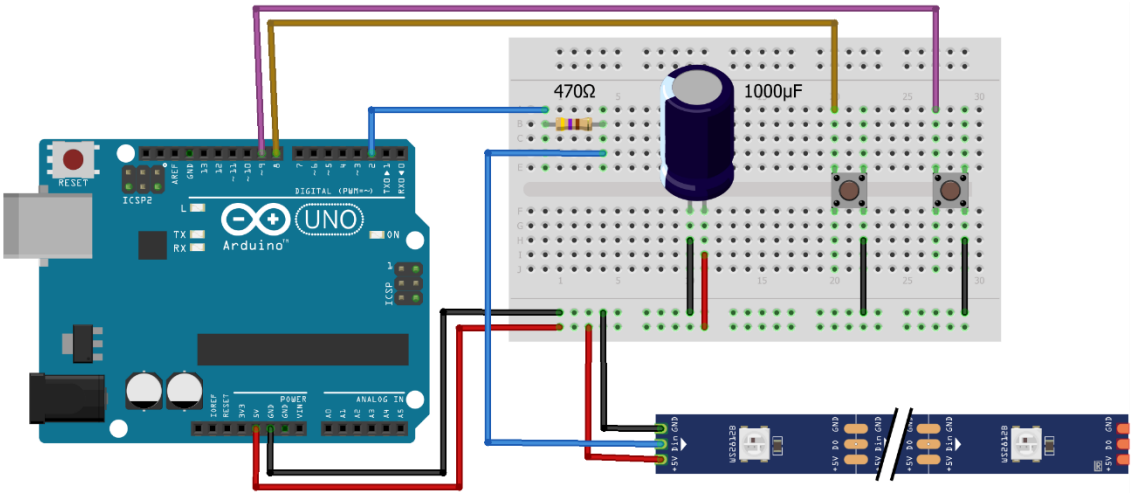
I si ho volem deixar tot més entenedor, podem definir constants pels valors numèrics que hem anat posant:

```
#define PIN_SENYAL_SO A0
#define OFFSET_SO 514
#define MAXIM_SO 560
#define MITJANA_LECTURES 50
```

Canvieu el valor de la constant ESPERA de 4 a 2 per donar-li més agilitat i jugueu amb sons d'instruments purs https://www.youtube.com/watch?v=to73mH1MTwc

### 4.6.3) LED Race

Per fer una carrera de LEDs necessitem tenir sols dos botons que al prémer-los ens vagin avançant les posicions de cadascun dels jugadors. Utilitzarem aquest circuit:




#### 4.6.3.1) LED Race Atrapar-se

Farem una primera versió més senzilla però molt divertida, a on aprendrem els blocs que compondran el codi del joc. Tindrem dos jugadors colocats en extrems oposats, que avançaran una posició a cada polsació, i guanyarà el primer que atrapi a l'altre. A cada volta completa que fagin, la longitud del jugador creixerà, el que farà que sigui més senzill atrapar-se.

Comencem amb el que ja coneixem, incloent la llibreria, anomenant tots els pins que fem servir, el nombre de leds de la tira i un retard per donar-li una certa realitat al moviment. Definim l'array de la tira i totes les variables que farem servir per controlar els jugadors: les posicions a on estan, el nombre de voltes que porten i la posició final de la cua per controlar si s'han enxampat. Finalment necessitarem gestionar els polsadors: llegirem els botons, tindrem emmagatzemats l'estat anterior per veure si l'hem premut o despremut i una tercera per indicar que s'ha premut i cal fer una acció.

```
#include "FastLED.h"

#define PIN_TIRA_LED    2
#define NUM_LEDS       30
#define PIN_POLSADOR_1  8
#define PIN_POLSADOR_2  9
#define ESPERA         50

CRGB leds[NUM_LEDS];

int posPlayer1, posPlayer2;
int voltaPlayer1, voltaPlayer2;
int posCuaPlayer1, posCuaPlayer2;
int color;

boolean lecturaBoto1, estatBoto1, premutBoto1;
boolean lecturaBoto2, estatBoto2, premutBoto2;
```

Per estructurar millor el codi i simplificar la part principal, abans de començar amb el Setup i Loop ens caldrà definir una funció que fagi l'animació quan hi hagi un guanyador.

Senzillament li passem com a paràmetre el jugador que ha guanyat i, amb el mètode d'encendre el cap i apagar la cua, anem fent rotar un cuc amb la longitud del jugador que ha guanyat tot partint de la posició en què l'ha atrapat.

```
void winner(int player) {
   int cap, cua, longitud;

   if (player == 1) { cap = posPlayer1; longitud = voltaPlayer1; }
   if (player == 2) { cap = posPlayer2; longitud = voltaPlayer2; }
   while (true) {
      cua = cap-longitud;
      if (cua < 0) cua = NUM_LEDS + cua;
      if (player == 1) { leds[cap] = CRGB(0, 255, 0); leds[cua] = CRGB(0, 0, 0); }
      if (player == 2) { leds[cap] = CRGB(0, 0, 255); leds[cua] = CRGB(0, 0, 0); }
      FastLED.show();
      delay(ESPERA);
      cap++;
      if (cap >= NUM_LEDS) cap = 0;
   }
}
```

Ara ja sí que tenim la inicialització de la tira i totes les proteccions de seguretat que hem anat veient en altres apartats, la definició dels pins dels polsadors com a entrades digitals amb PullUp per estalviar-nos d'afegir unes resistències extra al circuit, i les inicialitzacions necessàries de certes variables que requereixen un valor per començar.

```
void setup() {
   FastLED.addLeds<NEOPIXEL, PIN_TIRA_LED> (leds, NUM_LEDS);
   FastLED.setMaxPowerInVoltsAndMilliamps(5, 500);
   FastLED.setBrightness(50);
   FastLED.clear(true);

   pinMode(PIN_POLSADOR_1, INPUT_PULLUP);
   pinMode(PIN_POLSADOR_2, INPUT_PULLUP);

   posPlayer1 = 0;
   posPlayer2 = NUM_LEDS / 2;
   voltaPlayer1 = 1;
   voltaPlayer2 = 1;

   estatBoto1 = HIGH;
   estatBoto2 = HIGH;
   premutBoto1 = false;
   premutBoto2 = false;
}
```

I arribem al cor del programa, a on marcarem els 4 passos de la dinàmica del joc que anirem repetint cíclicament fins que hi hagi un vencedor.

Llegirem el valor dels polsadors i si l'estat anterior és l'inrevés, haurem trobat que s'ha premut o despremut. Quan trobem que es prem ho memoritzarem a una variable per actuar a posteriori.

```
void loop() {
   // Lectura dels polsadors
   lecturaBoto1 = digitalRead(PIN_POLSADOR_1);
   if ((lecturaBoto1 == LOW) && (estatBoto1 == HIGH)) {    // Premut 1
      premutBoto1 = true;
      estatBoto1 = LOW;
   }
   if ((lecturaBoto1 == HIGH) && (estatBoto1 == LOW)) {    // Despremut 1
      estatBoto1 = HIGH;
   }
   lecturaBoto2 = digitalRead(PIN_POLSADOR_2);
   if ((lecturaBoto2 == LOW) && (estatBoto2 == HIGH)) {    // Premut 2
      premutBoto2 = true;
      estatBoto2 = LOW;
   }
   if ((lecturaBoto2 == HIGH) && (estatBoto2 == LOW)) {    // Despremut 2
      estatBoto2 = HIGH;
   }
```

Quan detectem que s'ha premut el botó actuarem. En aquest joc el que caldrà fer és avançar una posició el jugador. Afegirem el control de circuit tancat, és a dir, que quan arribem a l'extrem de la tira el fem anar al principi. I també controlem el comptatge de voltes completades. Finalment esborrarem l'indicador de polsació, tot esperant-ne una altra que el fagi avançar de nou.

```
// Acció al prémer els botons
if (premutBoto1 == true) {
   posPlayer1++;
   if (posPlayer1 >= NUM_LEDS) posPlayer1 = 0;
   if (posPlayer1 == 0) voltaPlayer1++;
   premutBoto1 = false;
}
if (premutBoto2 == true) {
   posPlayer2++;
   if (posPlayer2 >= NUM_LEDS) posPlayer2 = 0;
   if (posPlayer2 == NUM_LEDS/2) voltaPlayer2++;
   premutBoto2 = false;
}
```

Una vegada ja actualitzats els comptadors, caldrà visualitzar-ho. Esborrarem la tira i pintarem cada jugador amb la seva cua corresponent en color atenuant-se.

```
// Actualitza la posició dels jugadors
FastLED.clear();
for(int i=0; i<voltaPlayer1; i++) {
   color = 255-255*i*2/NUM_LEDS;
   if (posPlayer1-i >= 0) leds[posPlayer1-i] = CRGB(0, color, 0);
   else leds[NUM_LEDS+posPlayer1-i] = CRGB(0, color, 0);
}
for(int i=0; i<voltaPlayer2; i++) {
   color = 255-255*i*2/NUM_LEDS;
   if (posPlayer2-i >= 0) leds[posPlayer2-i] = CRGB(0, 0, color);
   else leds[NUM_LEDS+posPlayer2-i] = CRGB(0, 0, color);
}
FastLED.show();
```

I ja sols ens falta comprovar si algú ha atrapat la cua de l'altra. Calculem quina és l'última posició de la cua i fem una simple comparació. Si trobem un guanyador invoquem a la funció creada a l'inici per a que ens fagi l'efecte de victòria.

```
   // Analitza si algú ha guanyat
   posCuaPlayer1 = posPlayer1 - voltaPlayer1;
   if (posCuaPlayer1 < 0) posCuaPlayer1 = NUM_LEDS + posCuaPlayer1;
   posCuaPlayer2 = posPlayer2 - voltaPlayer2;
   if (posCuaPlayer2 < 0) posCuaPlayer2 = NUM_LEDS + posCuaPlayer2;
   if (posPlayer1 == posCuaPlayer2) winner(1);
   if (posPlayer2 == posCuaPlayer1) winner(2);
}
```

Sempre que es treballa amb polsadors es corre el risc que produeixin rebots, i pet tant falses deteccions d'incorrectes polsacions. És important modificar el codi de la lectura dels polsadors per afegir-li un Debounce, que és un període de temps en el qual no acceptem cap lectura més un cop detectat un primer canvi, ja sigui de premut o despremut, permetent al polsador que s'estabilitzi.

Farem 3 modificacions:

- Afegirem dues noves variables que necessitarem a l'inici del programa.

```
#define DEBOUNCE_DELAY 50
unsigned long debounceTime1, debounceTime2;
```

- Les inicialitzarem dins el Setup.

```
debounceTime1 = 0;
debounceTime2 = 0;
```

- I a l'inici del Loop canviarem lleugerament la lectura dels polsadors. Utilitzarem la funció millis() que retorna un nombre equivalent al temps que fa que s'ha resetejat l'Arduino, en

milisegons. Memoritzant aquest temps a les noves variables quan es detecta una primera transició, no permetrem accedir a llegir el polsador de nou fins que hagi passat el temps definit de retràs DEBOUNCE_DELAY.

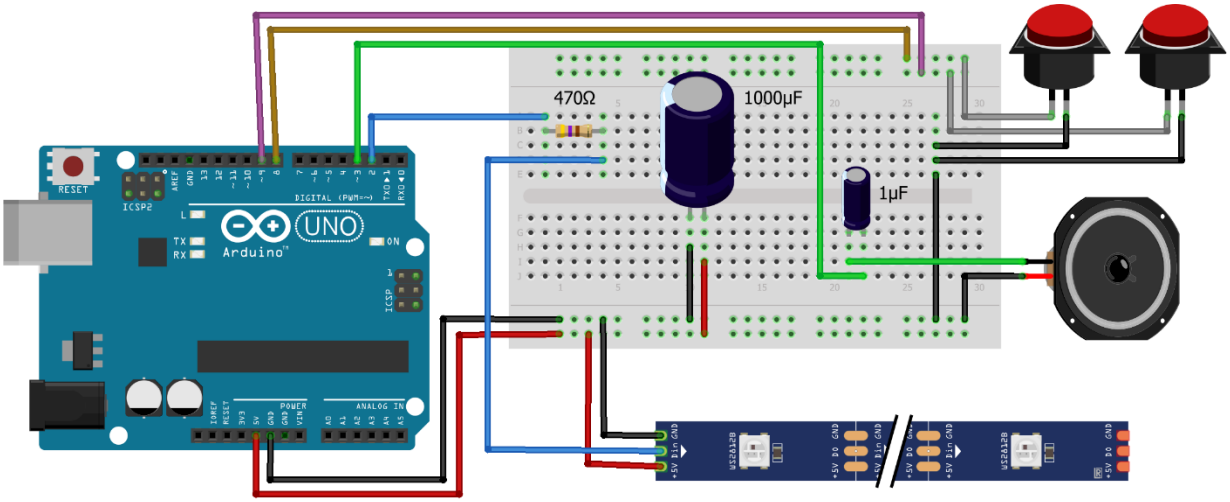
```
// Lectura dels polsadors amb supressió dels rebots
if ((millis() - debounceTime1) > DEBOUNCE_DELAY) {
   lecturaBoto1 = digitalRead(PIN_POLSADOR_1);
   if ((lecturaBoto1 == LOW) && (estatBoto1 == HIGH)) {    // Premut 1
      premutBoto1 = true;
      estatBoto1 = LOW;
      debounceTime1 = millis();
   }
   if ((lecturaBoto1 == HIGH) && (estatBoto1 == LOW)) {    // Despremut 1
      estatBoto1 = HIGH;
      debounceTime1 = millis();
   }
}
if ((millis() - debounceTime2) > DEBOUNCE_DELAY) {
   lecturaBoto2 = digitalRead(PIN_POLSADOR_2);
   if ((lecturaBoto2 == LOW) && (estatBoto2 == HIGH)) {    // Premut 2
      premutBoto2 = true;
      estatBoto2 = LOW;
      debounceTime2 = millis();
   }
   if ((lecturaBoto2 == HIGH) && (estatBoto2 == LOW)) {    // Despremut 2
      estatBoto2 = HIGH;
      debounceTime2 = millis();
   }
}
```

#### 4.6.3.1) LED Race Carrera a Voltes

La segona versió es molt més elaborada. Es tracta d'un Scalextric o Slot, a on tots dos jugadors parteixen d'una mateixa línia de sortida i guanya qui completa abans un nombre definit de voltes a tot el circuit, podent-se anar avançant. Afegeix cinètica per a simular arrencada i frenada, pujades i baixades, i inclús loopings, un semàfor que marcarà l'inici de la cursa, i sons de motor, d'acceleració, d'avançament i de pas per meta per fer-lo bastant realista, i una sorpresa de música per coronar al guanyador.

Es tracta d'un magnífic projecte open-source desenvolupat per [https://openledrace.net/](https://openledrace.net/) que hem agafat i adaptat a les nostres necessitats.

Utilitzarem un pin de sortida addicional per a generar el so, amb un petit altaveu i un condensador en sèrie, que la seva funció serà no permetre que el corrent continu passi a l'altaveu i pugui cremar la sortida GPIO. Els polsadors ja seran comanaments "gamers" que construirem per a l'ocasió.



```
#include <FastLED.h>
#define MAXLEDS   90   // MAX LEDs actives on strip

#define PIN_LED    2   // R 500 ohms to data pin, CAP 1000 uF to VCC_5v/GND, Power Supply 5V 2A
#define PIN_P1     8   // switch player 1 to PIN and GND
#define PIN_P2     9   // switch player 2 to PIN and GND
#define PIN_AUDIO  3   // through Capacitor (1 to 10uf) to speaker 8 ohms

// definició de la rampa
#define INI_RAMP   50
#define MED_RAMP   60
#define END_RAMP   70
#define HIGH_RAMP  6

CRGB leds[MAXLEDS];
byte gravity_map[MAXLEDS];

float ACEL=0.1;   // acceleration value (0.1 for single strip density & 0.2 for double)
float kf=0.015;   // friction constant
float kg=0.003;   // gravity constant

#define COLOR1   CRGB(0,255,0)
#define COLOR2   CRGB(0,0,255)
```

```
byte flag_sw1=0;
byte flag_sw2=0;

float speed1=0;
float speed2=0;
float dist1=0;
float dist2=0;
byte loop1=0;
byte loop2=0;

byte loop_max=5;   // total race laps
byte leader=0;
byte draworder=0;
#define TDELAY 5

int TBEEP=0;
int FBEEP=0;
```

```
void set_ramp(byte H, byte a, byte b, byte c) {
   for(int i=0; i<(b-a); i++) { gravity_map[a+i]=127-i*((float)H/(b-a)); }
   gravity_map[b]=127;
   for(int i=0; i<(c-b); i++) { gravity_map[b+i+1]=127+H-i*((float)H/(c-b)); }
}

void set_loop(byte H, byte a, byte b, byte c) {
   for(int i=0; i<(b-a); i++) { gravity_map[a+i]=127-i*((float)H/(b-a)); }
   gravity_map[b]=255;
   for(int i=0; i<(c-b); i++) { gravity_map[b+i+1]=127+H-i*((float)H/(c-b)); }
}
```

```
void setup() {
   FastLED.addLeds<NEOPIXEL, PIN_LED> (leds, MAXLEDS);
   FastLED.setMaxPowerInVoltsAndMilliamps(5, 500);
   FastLED.setBrightness(50);
   FastLED.clear(true);
   pinMode(PIN_P1, INPUT_PULLUP);
   pinMode(PIN_P2, INPUT_PULLUP);

   // defineix la cinètica a cada posició (127 pla, <127 pujada, >127 baixada)
   for(int i=0; i<MAXLEDS; i++) { gravity_map[i]=127; }
   set_ramp(HIGH_RAMP, INI_RAMP, MED_RAMP, END_RAMP);

   start_race();
}
```

```
void start_race() {
   // mostra semàfor i fa sonar l'inici de cursa
   FastLED.clear(true);
   delay(2000);
   leds[9]=CRGB(255,0,0);
   leds[8]=CRGB(255,0,0);
   FastLED.show();
   tone(PIN_AUDIO,294);  delay(2000);  noTone(PIN_AUDIO);   // Preparats: Re 4a Octava
   leds[9]=CRGB(0,0,0);
   leds[8]=CRGB(0,0,0);
   leds[7]=CRGB(255,255,0);
   leds[6]=CRGB(255,255,0);
   FastLED.show();
   tone(PIN_AUDIO,587);  delay(2000);  noTone(PIN_AUDIO);   // Llestos: Re 5a Octava
   leds[7]=CRGB(0,0,0);
   leds[6]=CRGB(0,0,0);
   leds[5]=CRGB(0,255,0);
   leds[4]=CRGB(0,255,0);
   FastLED.show();
   tone(PIN_AUDIO,1174);  delay(2000);  noTone(PIN_AUDIO);   // Ja!!!: Re 6a Octava

   speed1=0;  speed2=0;
   dist1=0;  dist2=0;
   loop1=0;  loop2=0;
}
```

```
void winner_fx(byte w) {
   int win_intro[] = {659, 659, 0, 659, 0, 523, 659, 0, 784, 0, 0, 0, 392, 0, 0, 0};
   int win_music[] = {523, 0, 0, 392, 0, 0, 330, 0, 0, 440, 0, 494, 466, 440, 0,
                      392, 659, 784, 880, 0, 698, 784, 0, 659, 0, 523, 587, 494, 0, 0};

   FastLED.clear();
   if (w==1) for(int i=0; i<MAXLEDS/4; i++) { leds[i]=COLOR1; }
   if (w==2) for(int i=0; i<MAXLEDS/4; i++) { leds[i]=COLOR2; }
   FastLED.show();
   int msize = sizeof(win_intro) / sizeof(int);
   for(int note=0; note<msize; note++) {
      tone(PIN_AUDIO, win_intro[note], 150);
      delay(180);
   }
   msize = sizeof(win_music) / sizeof(int);
   while(true) {
      for(int note=0; note<msize; note++) {
         tone(PIN_AUDIO, win_music[note], 150);
         delay(180);
      }
   }
}
```

```
void draw_car1(void) {
   int index;
   for(int i=0; i<=loop1; i++) {
      index=((word)dist1 % MAXLEDS)+i;
      if (index>=MAXLEDS) index-=MAXLEDS;
      leds[index]=COLOR1;
   }
}

void draw_car2(void) {
   int index;
   for(int i=0; i<=loop2; i++) {
      index=((word)dist2 % MAXLEDS)+i;
      if (index>=MAXLEDS) index-=MAXLEDS;
      leds[index]=COLOR2;
   }
}
```

```
void loop() {
   // esborra la tira led
   FastLED.clear();

   // mostra la rampa
   int value;
   for(int i=0; i<(MED_RAMP-INI_RAMP); i++) {
      value=(i+1)*255/(MED_RAMP-INI_RAMP);
      leds[INI_RAMP+i]=CRGB(value,0,value);
   }
   for(int i=0; i<(END_RAMP-MED_RAMP); i++) {
      value=(i+1)*255/(END_RAMP-MED_RAMP);
      leds[END_RAMP-i]=CRGB(value,0,value);
   }
```

```
   // si Player1 ha premut incrementa la velocitat
   if ((flag_sw1==1) && (digitalRead(PIN_P1)==0)) { flag_sw1=0;  speed1+=ACEL; }
   if ((flag_sw1==0) && (digitalRead(PIN_P1)==1)) flag_sw1=1;

   // aplica la constant de gravetat per frenar/accelerar a la pujada/baixada de la rampa
   if ((gravity_map[(word)dist1 % MAXLEDS])<127)
      speed1-=kg*(127-(gravity_map[(word)dist1 % MAXLEDS]));
   if ((gravity_map[(word)dist1 % MAXLEDS])>127)
      speed1+=kg*((gravity_map[(word)dist1 % MAXLEDS])-127);

   // aplica la constant de la fricció per frenar-lo
   speed1-=speed1*kf;
```

```
   // idem per Player2
   if ((flag_sw2==1) && (digitalRead(PIN_P2)==0)) { flag_sw2=0;  speed2+=ACEL; }
   if ((flag_sw2==0) && (digitalRead(PIN_P2)==1)) flag_sw2=1;

   if ((gravity_map[(word)dist2 % MAXLEDS])<127)
      speed2-=kg*(127-(gravity_map[(word)dist2 % MAXLEDS]));
   if ((gravity_map[(word)dist2 % MAXLEDS])>127)
      speed2+=kg*((gravity_map[(word)dist2 % MAXLEDS])-127);

   speed2-=speed2*kf;
```

```
   // actualitza la distancia recorreguda
   dist1+=speed1;
   dist2+=speed2;

   // comprova si hi ha canvi de líder i prepara per emetre un so addicional
   if (dist1>dist2) {
      if (leader==2) { FBEEP=440;  TBEEP=10; }
      leader=1;
   }
   if (dist2>dist1) {
      if (leader==1) { FBEEP=440*2;  TBEEP=10; }
      leader=2;
   }

   // al passar per l'inici incrementa el comptador de voltes i també emet so
   if (dist1>MAXLEDS*loop1) { loop1++;  FBEEP=440;  TBEEP=10; }
   if (dist2>MAXLEDS*loop2) { loop2++;  FBEEP=440*2;  TBEEP=10; }
```

```
   // si Player1 o Player2 han fet les voltes ja tenim guanyador
   if (loop1>loop_max) winner_fx(1);
   if (loop2>loop_max) winner_fx(2);
```

```
    // alterna l'ordre de visualització dels cotxes (per si comparteixen caselles)
    // i els pinta
    if ((millis() & 512)==(512*draworder)) {
       if (draworder==0) draworder=1;
       else draworder=0;
    }
    if (abs(round(speed1*100))>abs(round(speed2*100))) draworder=1;
    if (abs(round(speed2*100))>abs(round(speed1*100))) draworder=0;

    if ( draworder==0 ) {
       draw_car1();
       draw_car2();
    }
    else {
       draw_car2();
       draw_car1();
    }
    FastLED.show();
```

```
    // so del motor dels cotxes en funció de la velocitat i els sons addicionals
    tone(PIN_AUDIO,FBEEP+int(speed1*440*2)+int(speed2*440*3));
    if (TBEEP>0) TBEEP--;
    else FBEEP=0;

    delay(TDELAY);
}
```

### 4.6.4) Twang

https://github.com/Critters/TWANG
https://blog.arduino.cc/2018/01/24/crawl-through-a-1d-led-dungeon-with-twang/
https://bavatuesdays.com/building-a-line-wobbler-clone-with-twang/
https://www.buildlog.net/blog/2018/01/twang/