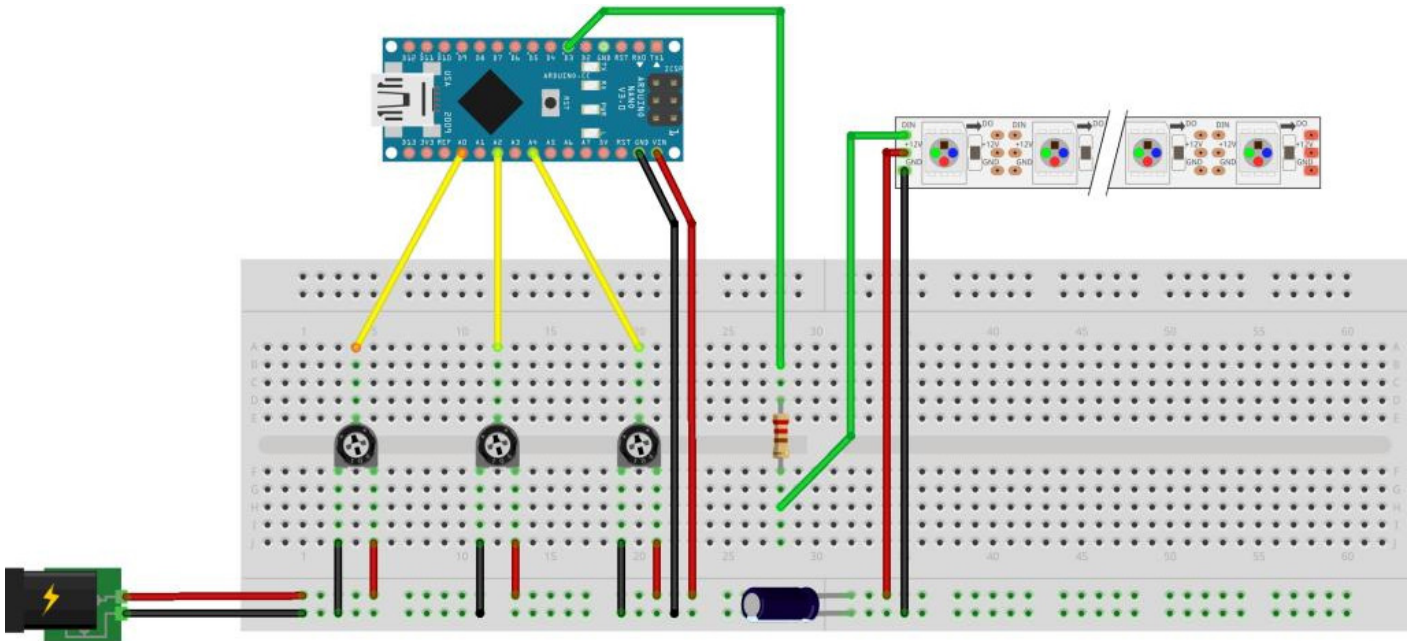


Construction

- [Électronique](#)
- [Programme arduino](#)

Électronique

Schéma



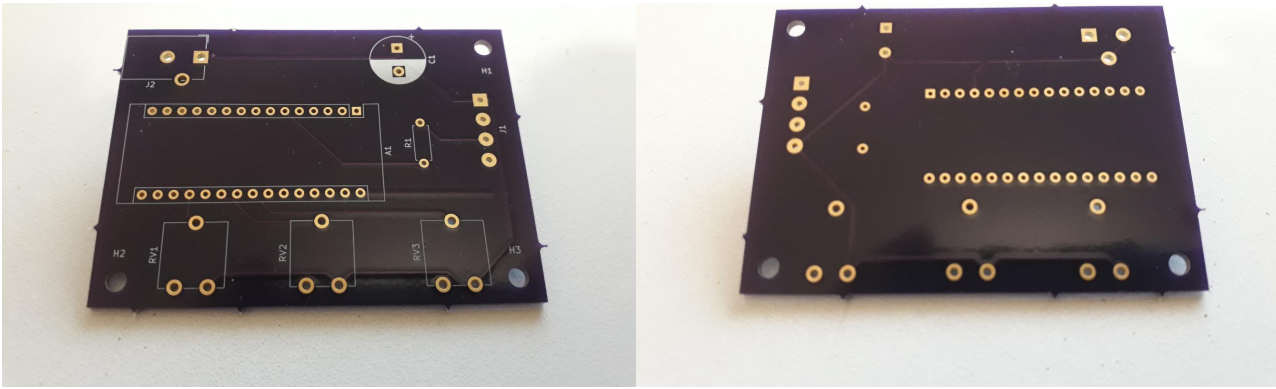
Les 3 potentiomètres servent de contrôle : leurs valeurs mesurées par l'arduino vont servir à contrôler la bande LED (par exemple sa couleur). Le condensateur sert de sécurité : lors de la variation d'un potentiomètre, ou de l'allumage des LEDs, il va absorber les chutes ou pics de tension.

(contrairement à l'illustration, la bande LED est alimentée en 5V, et non en 12V)

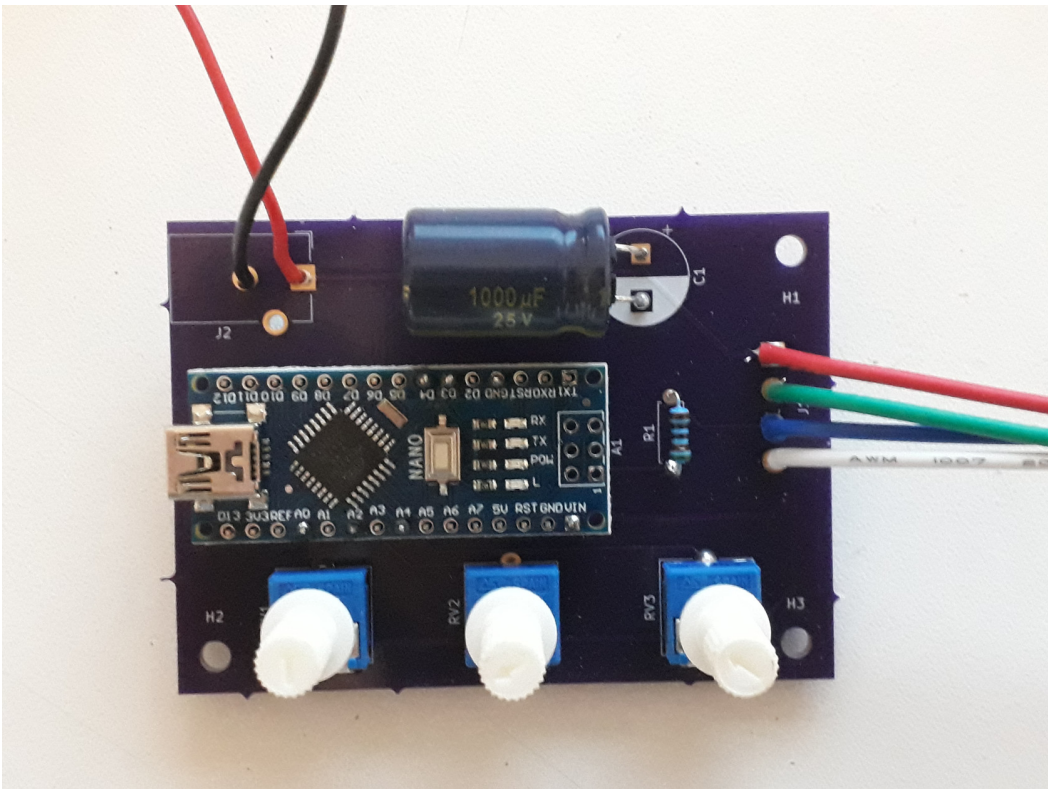
Création d'un circuit imprimé (PCB) et assemblage

L'ensemble des composants est assemblé sur un PCB, conçu avec [KiCad](#), et commandé sur [OshPark](#) (il m'en reste deux exemplaires, contactez-moi si vous en voulez un). Fichiers de projet KiCad (et Gerber pour la production) : [SEP_LEDS.zip](#)

Le PCB :



Après soudure des composants :



Remarque : les bandes LED usuelles ont 3 fils : GND, +5V et data. Ici j'ai utilisé une bande LED WS2813, qui possède 4 fils : GND, +5V et 2 fils data. Pour l'utiliser avec la bibliothèque Fastled de l'arduino, j'ai connecté les deux fils data ensemble.

Programme arduino

Le programme combine deux parties : la gestion de la bande LED (et interactions avec les potentiomètres), et la simulation du modèle physique. Les deux sont mélangés tout au long du code (téléchargeable ici : [FastLED_SEP_pot_biais.ino](#)), mais pour simplifier et illustrer d'autres exemples d'utilisation du circuit, je vais d'abord décrire la gestion des LEDs uniquement. La description du modèle physique et le programme correspondant sont donnés plus bas.

Gestion des LEDs et des potentiomètres

Pour illustrer le fonctionnement de base, on va écrire un programme qui va allumer toutes les LEDs de la même couleur. Celle-ci est déterminée par les 3 potentiomètres (gauche -> rouge, centre -> vert, droite -> bleu).

```
#include <FastLED.h>

// nombre de LEDs
#define NUM_LEDS 144

// pin sur lequel est connecté le fil data
#define DATA_PIN 3

// intensité des LEDs
#define BRIGHTNESS 20

// pin pour le potentiomètre de gauche
const int PinG = A0;
// valeur du potentiomètre
int potPinG ;

// pin pour le potentiomètre central
const int PinC = A2;
// valeur du potentiomètre
int potPinC ;

// pin pour le potentiomètre de droit
const int PinD = A4;
// valeur du potentiomètre
```

```
int potPinD ;

// tableau des couleurs de chaque LED
CRGB leds[NUM_LEDS];

// initialisation
void setup() {

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

    // limitation de la puissance, via voltage max (5V) et courant max (1A)
    FastLED.setMaxPowerInVoltsAndMilliamps(5,1000);

    // ajuster la luminosité
    FastLED.setBrightness(BRIGHTNESS);
}

void loop() {

    // mesure des valeurs des potentiomètres
    potPinG = analogRead(PinG) ;
    potPinC = analogRead(PinC) ;
    potPinD = analogRead(PinD) ;

    // Conversion en valeurs entre 0 et 255
    int valR = map(potPinG, 0, 1023, 0, 255) ;
    int valG = map(potPinC, 0, 1023, 0, 255) ;
    int valB = map(potPinD, 0, 1023, 0, 255) ;

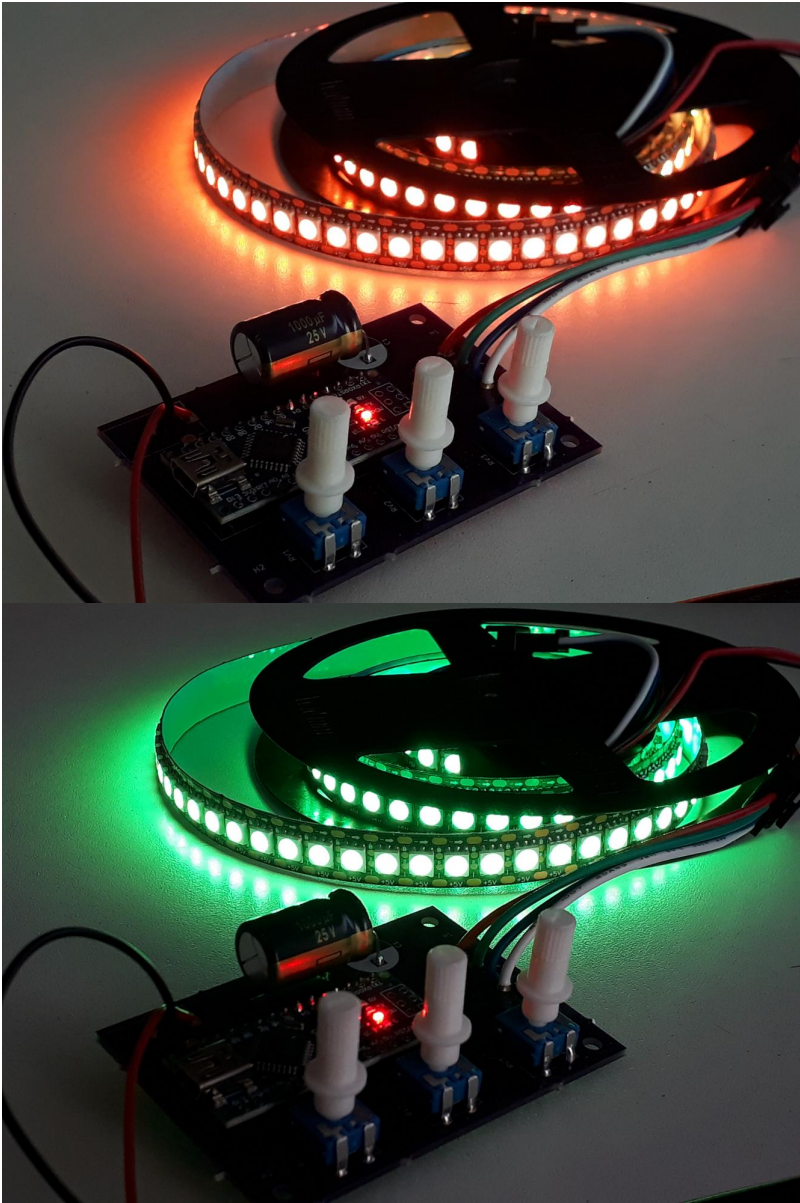
    // allumer les LEDs de la couleur R, G, B
    for(int i=0; i<NUM_LEDS; i++)
    {
        leds[i] = CRGB(valR, valG, valB);
    }

    // allumer les LEDs
    FastLED.show();

    // attendre 100ms
    delay(100) ;
```

```
}
```

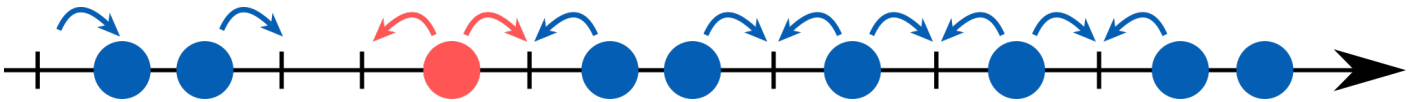
Voilà ce que ça donne après transfert sur l'Arduino :



Simulation du modèle

Le processus d'exclusion symétrique est un modèle de particules sur réseau. À chaque pas de temps, une particule est choisie au hasard. Celle-ci va alors tenter de sauter sur un site voisin, soit à gauche, soit à droite, avec la même probabilité. Le saut n'a lieu que si le site choisi est vide, sinon il ne se passe rien. Cela impose qu'il y a au maximum une particule par site. Une des particules est spécifique (en rouge ci-dessous) : lorsqu'elle est choisie, elle peut sauter à droite avec une probabilité pr , et vers la gauche avec une probabilité $1-pr$. En choisissant $pr > 1/2$, cette particule va avoir tendance à plutôt aller vers la droite que vers la gauche. Cela modélise l'action

d'une force agissant sur cette particule uniquement.



Initialement, la particule rouge est placée au milieu. Les autres particules sont placées aléatoirement avec la règle suivante :

- à gauche de la particule rouge, pour chaque site, on choisit aléatoirement si le site est occupé par une particule ou non. On note ρ_{om} cette probabilité;
- à droite, on procède de même, mais avec une probabilité différente, que l'on note ρ_{op} .

On va illustrer ce modèle à l'aide d'une bande LED. Les sites du réseau correspondent aux LEDs. Une LED allumée correspond à un site occupé par une particule. Les trois potentiomètres permettent de régler les paramètres du modèle :

- le potentiomètre de gauche contrôle la probabilité d'occupation à gauche ρ_{om} ;
- le potentiomètre central contrôle la probabilité de saut p_r de la particule rouge;
- le potentiomètre de droite contrôle la probabilité d'occupation à droite ρ_{op} .

```
// les bibliothèques nécessaires
#include <FastLED.h>
#include <math.h>

// nombre de LEDs = nombre de sites
#define NUM_LEDS 144
// pin sur lequel est connecté le fil data
#define DATA_PIN 3

// intensité des LEDs
#define BRIGHTNESS 20

// pour la génération de nombres aléatoires
// on va générer des entiers entre 0 et MAX_RANDOM
#define MAX_RANDOM 1000

// densités initiales, > 1 pour l'initialisation (voir plus bas)
double rho_m = 1.1 ;
double rho_p = 1.1 ;

// probabilité pour la particule centrale (rouge) de sauter à droite
double p_r = 0.5 ;
```

```

// pin pour la densité à gauche (potentiomètre gauche)
const int PinRhoL = A0;
int potPinRhoL ;

// pin pour la densité à droite (potentiomètre droit)
const int PinRhoR = A4;
int potPinRhoR ;

// pin pour le biais = proba d'aller à droite (potentiomètre central)
const int PinBias = A2;
int potPinBias ;

// tableau des couleurs de chaque LED
CRGB leds[NUM_LEDS];

// on va représenter l'état du système à un instant par des nombres d'occupations
// à chaque site est associé un nombre d'occupation :
// - 0 => le site est vide
// - 1 => le site contient une particule
// on considère que la particule rouge n'occupe pas de site (elle est traitée à part)
int occup[NUM_LEDS];

// position initiale de la particule rouge = au milieu
int pos_tracer = NUM_LEDS/2;

// nb de particles
// (avant l'initialisation, il n'y a aucune particule)
int npart = 0 ;

// on crée aussi un tableau qui contient les positions des particules
int pospart[NUM_LEDS] ;

// fonction qui génère des nombres aléatoires entre 0 et 1 à partir de la génération d'entiers
double Unif(){

    return random(MAX_RANDOM)/(MAX_RANDOM+1.);
}

// fonction auxiliaire qui initialise tout

```



```

void initAll() {

    // on a initialement aucune particule bleu
    npart = 0 ;
    // la particule rouge est au milieu
    pos_tracer = NUM_LEDS/2;

    // pour chaque site à gauche, on tire au hasard pour savoir s'il est occupé ou pas
    for(int i=0; i<pos_tracer; i++)
    {
        if(Unif() < rhom){
            occup[i] = 1 ;
        }
        else{
            occup[i] = 0 ;
        }
        // si le site est occupé, on illumine la LED
        leds[i] = CHSV(117, 255*occup[i], 127*occup[i]);
        // CHSV définit la couleur à partir de 3 paramètres (hue, saturation, valeur)
        // en mettant les deux derniers à zéro, on a du noir (LED éteinte)
        // on multiplie donc ces nombres par le nombre d'occupation : si le site est vide, ça fait zéro donc éteint !

        // on se rappelle qu'on a une particule au site i
        pospart[npart] = i ;

        // on augmente le nombre total de particules si on en a placé une
        npart += occup[i] ;

    }

    // on s'occupe maintenant de la particule rouge

    // on stocke sa position
    pospart[npart] = pos_tracer ;
    // on a une particule de plus (la rouge)
    npart += 1 ;
    // on considère que le site n'est pas occupé par une particule bleue
    occup[pos_tracer] = 0 ;

    // on place les particules à droite de la particule rouge

```

```

// c'est tout comme avant, sauf que la probabilité d'avoir une particule sur un site est rhop
for(int i=pos_tracer+1; i<NUM_LEDS; i++)
{
    if(Unif() < rhop){
        occup[i] = 1 ;
    }
    else{
        occup[i] = 0 ;
    }

    leds[i] = CHSV(117, 255*occup[i], 127*occup[i]);

    pospart[npart] = i ;

    npart += occup[i] ;
}

// on allume la particule rouge
leds[pos_tracer] = CHSV(0, 255, 127);

// on affiche tout sur la bande LED
FastLED.show();

}

// lors de l'allumage de l'arduino
void setup() {

    // on initialise la gestion des LEDs
    FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUM_LEDS);

    // on limite la puissance en indiquant la tension max (5V), et le courant max (1A)
    // à adapter en fonction de l'alimentation branchée
    FastLED.setMaxPowerInVoltsAndMilliamps(5,1000);

    // on ajuste la luminosité brightness
    FastLED.setBrightness(BRIGHTNESS);

    // on initialise le générateur de nombre aléatoire à partir d'une mesure (bruitée donc aléatoire) du pin 0
    randomSeed(analogRead(0));
}

```

```

}

// la boucle de l'arduino
// à chaque étape, on vérifie si un la valeur d'un potentiomètre a été changée
// - si oui : on change les paramètres du modèle et on ré-initialise si besoin
// - si non : on essaye de bouger une particule
void loop() {

    // on lit la valeur du potentiomètre gauche
    potPinRhoL = analogRead(PinRhoL) ;
    // on convertit ça en nombre entre 0 et 1, par pas de 0.1 (0, 0.1, 0.2 ...)
    double RhoL = map(potPinRhoL, 0, 1023, 0, 10)/10. ;

    // pareil pour le potentiomètre droit
    potPinRhoR = analogRead(PinRhoR) ;
    double RhoR = map(potPinRhoR, 0, 1023, 0, 10)/10. ;

    // pour celui du milieu on convertit ça entre 0 et 0.5
    potPinBias = analogRead(PinBias) ;
    double Bias = map(potPinBias, 0, 1023, 0, 5)/10. ;
    // on ajoute cela à 0.5, ce qui donne une proba entre 0.5 et 1
    pr = 0.5 + Bias ;

    // s'il y a eu un changement des probas d'occupations initiales de chaque côte
    // alors on réinitialise tout, avec ces nouvelles probas
    if( abs(RhoL - rhom) > 0.01 || abs(RhoR - rhop) > 0.01 ){
        rhom = RhoL ;
        rhop = RhoR ;
        initAll() ;
    }

    // on ne met pas de temps d'attente
    // le temps de calcul est déjà suffisant
    delay(0) ;

    // on tire au hasard une particule, et on note sa position

```

```

int part = random(npart);
int pos = pospart[part] ;
// on va essayer de la déplacer vers new_pos
int new_pos ;

// la procédure dépend de si on a tiré la particule rouge ou pas
// si oui
if(pos == pos_tracer)
{
    // on calcule la nouvelle position
    // par défaut, on va à gauche (attention, si on sort par la gauche, on place la particule à droite)
    new_pos = (pos_tracer - 1 + NUM_LEDS) % NUM_LEDS ;
    // si on va à droite
    if(Unif() < pr){
        // on calcule la position correspondante
        new_pos = (pos_tracer + 1) % NUM_LEDS ;
    }

    // on regarde si le site d'arrivée est vide
    // si oui, on déplace la particule
    if(occup[new_pos] == 0)
    {
        // on éteint la LED de l'ancienne position
        leds[pos_tracer] = CRGB::Black;
        // on allume la nouvelle en rouge
        leds[new_pos] = CHSV(0, 255, 127);

        // on stocke les nouvelles positions
        pos_tracer = new_pos ;
        pospart[part] = new_pos ;
    }
}

// si c'est une particule bleue
else
{
    // on choisit le nouveau site aléatoirement, à gauche ou à droite avec la même proba
    new_pos = (pos + (2*random(2) - 1) + NUM_LEDS) % NUM_LEDS ;

    // on vérifie que le site d'arrivée est vide
    if(occup[new_pos] == 0 && pos_tracer != new_pos)

```

```
{

    // on éteint l'ancienne position
    leds[pos] = CRGB::Black;
    // on allume la nouvelle en bleu
    leds[new_pos] = CHSV(117, 255, 127);

    // on met à jour les nombres d'occupations
    occup[pos] = 0 ;
    occup[new_pos] = 1 ;

    // et la position
    pospart[part] = new_pos ;
}
}

// on affiche sur les LEDs
FastLED.show();

}
```