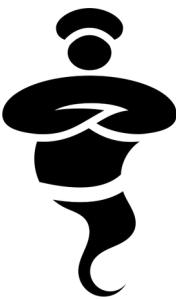


El Fene Aladin		CIM 2020-2021
	Projet Simon	vendredi 7 mai 2021

Compte rendu complet

Table des matières

1 But du projet.....	2
1.1 BDD.....	2
Simon Original.....	2
Nouveau Simon.....	3
1.2 IBD.....	3
2 Schéma stucturel.....	4
2.1 BOM.....	4
2.2 ERC.....	5
2.3 Calcul du coût de la carte.....	5
3 Routage de la carte.....	5
3.1 DRC.....	5
3.2 Remarques.....	6
3.3 Pistes cotés top.....	6
3.4 Pistes cotés bottom.....	7
3.5 Composants cotés top.....	7
3.6 Composants cotés bottom.....	8
3.7 Correction.....	8
3.8 Fabrication.....	9
Une première tentative.....	9
Préparation de la deuxième carte.....	9
Fabrication de la deuxième carte.....	10
3.9 Boîtier.....	11
4 Programme de validation.....	13
4.1 Les NeoPixels.....	13
Comportement de base.....	14
Écriture d'une fonction.....	14
4.2 Les boutons poussoirs.....	15
Comportement de base.....	15
Petit défaut personnel.....	17
Combinaison des NeoPixels et des boutons poussoirs.....	19
4.3 Le Buzzer.....	21
Comportement de base.....	21
Combinaison boutons poussoirs et NeoPixels.....	22
4.4 Presque un Simon !.....	24
5 Algorithme général.....	27



1 But du projet

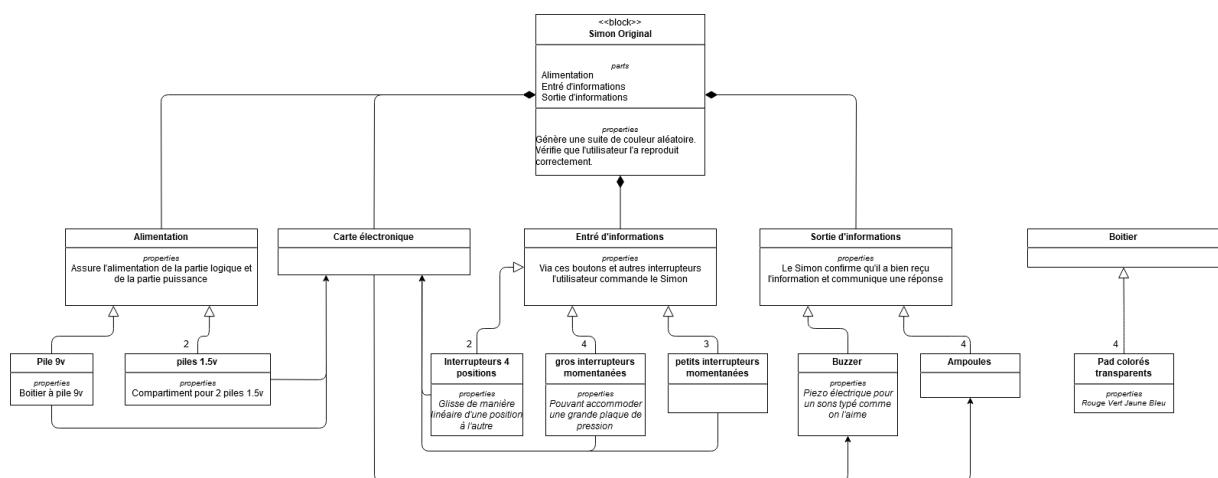
Le mini projet Simon va nous permettre de mieux appréhender les différentes étapes menant à la création d'un produit. Analyser le cahier des charges, s'inspirer de l'existant, produire des diagrammes cohérents pour nous aider par la suite à concevoir un produit répondant aux attentes. De plus, le circuit électronique conçu devra être inclus dans un boîtier imprimé 3d. Il faut donc bien prendre en compte le positionnement des différents composants.

Également, nous gagnerons en autonomie lors de la fabrication des circuits, c'est essentiel pour itérer rapidement dans le futur.

La programmation d'un système complet sera elle aussi très enrichissante, le potentiel pédagogique de ce projet est élevé.

1.1 BDD

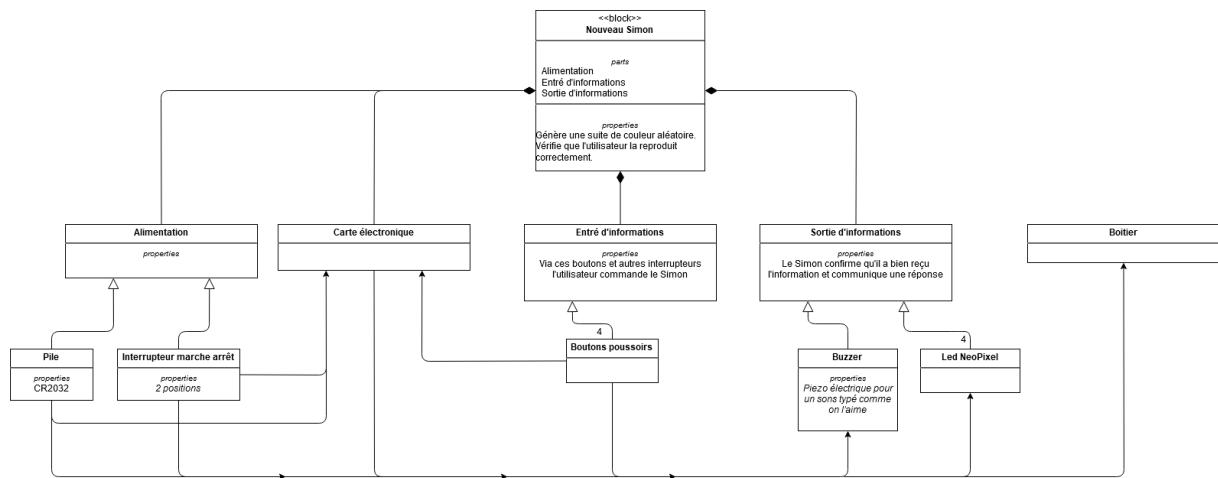
Simon Original



Le BDD image les connections entre les différents composants de l'objet. J'ai par ailleurs trouvé [un excellent article](#) détaillant la rétro ingénierie des Simons originaux.



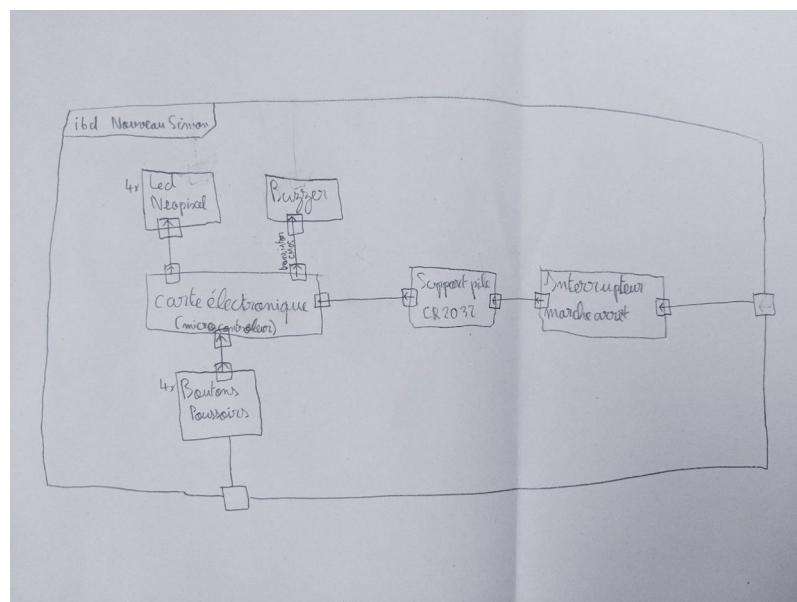
Nouveau Simon



Le nouveau Simon est très similaire à l'original, quoi qu'un peu simplifié. J'envisage un format compact et autonome, pour jouer n'importe où. Il pourra se faire discret grâce à un mode muet. La luminosité des led pourrait également être réglable.

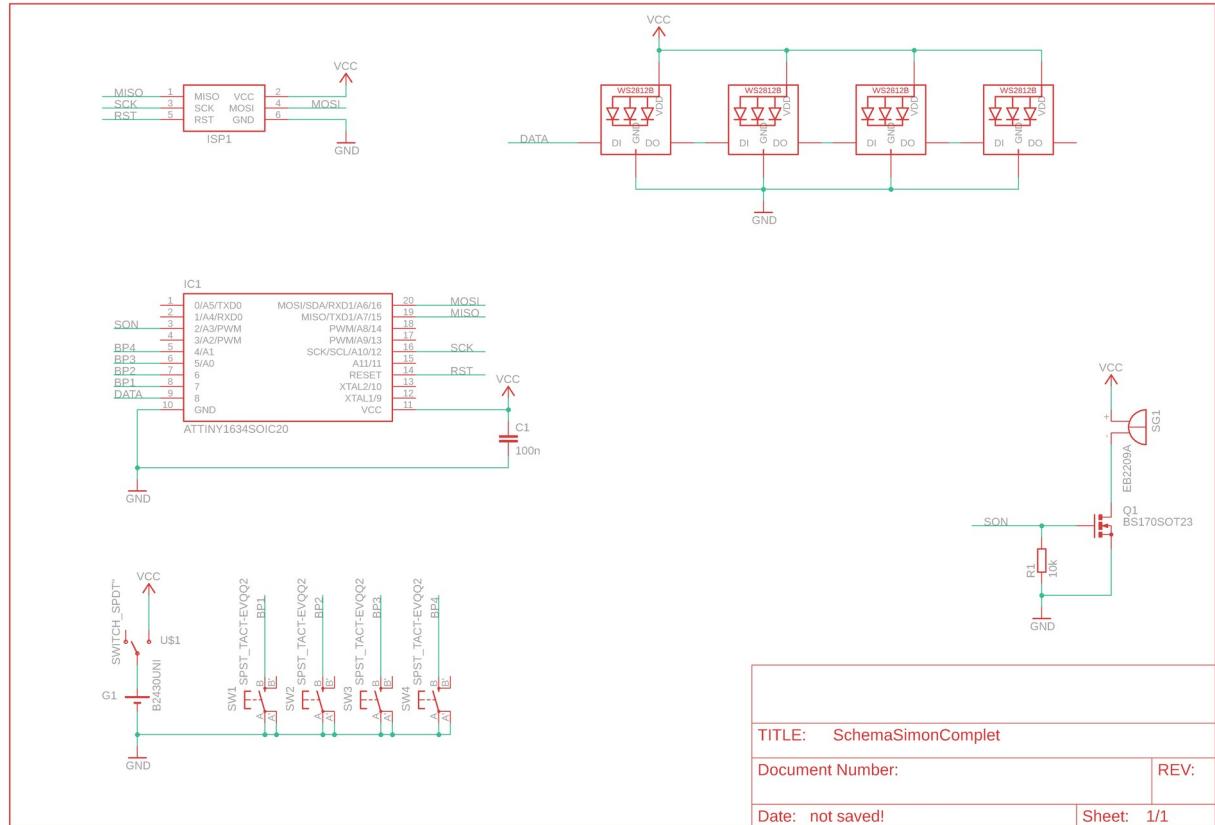
Ses 4 leds neopixel multicolore permettront de chouettes animations pour encourager l'utilisateur à continuer de jouer. De plus, ce sont les leds qui indiqueront la couleurs des boutons. Il est ainsi possible changer l'emplacement des couleurs pour plus de difficulté. (cela combat la mémoire musculaire qui se met en place à force de jouer).

1.2 IBD



L'ibd détail les moyens d'interaction externe et les transferts d'information interne à l'objet. Peux de composants sont nécessaires, la communication entre-elle reste rudimentaire.

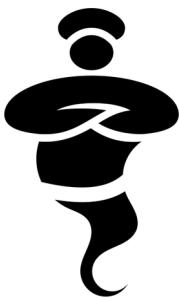
2 Schéma stucturel



Il a fallut choisir astucieusement les broches à utiliser pour connecter chaque composant en fonction de ses besoins.

2.1 BOM

Nom	Valeur	Référence	Boîtier	Description
C1	100n	C-EUC1206	C1206	Condensateur
G1	B2430UNI	B2430UNI	B2430UNI	Porte pile
IC1	ATTINY1634SOIC20	ATTINY1634SOIC20	SOIC20	Micro-controlleur
ISP1	AVRISP-6	AVRISP-6	AVRISP	Connecteur ISP AVR
LED1	WS2812B5050	WS2812B5050	WS2812B	Led adressable RGB
LED2	WS2812B5050	WS2812B5050	WS2812B	Led adressable RGB
LED3	WS2812B5050	WS2812B5050	WS2812B	Led adressable RGB
LED4	WS2812B5050	WS2812B5050	WS2812B	Led adressable RGB
Q1	BS170SOT23	BS170SOT23	SOT23	Mosfet canal N
R1	10k	R-EUR1206	R1206	Résistance
SG1	EB2209A	EB2209A	EB2209A	Buzzer de chez Buerklin
SW1	SPST_TACT-EVQQ2	SPST_TACT-EVQQ2	EVQ-Q2	Intérupteur SMT 6mm série EVQQ2
SW2	SPST_TACT-EVQQ2	SPST_TACT-EVQQ2	EVQ-Q2	Intérupteur SMT 6mm série EVQQ2
SW3	SPST_TACT-EVQQ2	SPST_TACT-EVQQ2	EVQ-Q2	Intérupteur SMT 6mm série EVQQ2
SW4	SPST_TACT-EVQQ2	SPST_TACT-EVQQ2	EVQ-Q2	Intérupteur SMT 6mm série EVQQ2
U\$1	SWITCH_SPDT","SWITCH_SPDT","CMS-CL-SB-12A-02T	SWCH-10651		Intérupteur marche arrêt



2.2 ERC

Type

- Consistency not checked (no board load...)
- Errors (0)
- Warnings (0)
- Approved (5)
 - POWER pin LED1 VDD connected to ... 1
 - POWER pin LED2 VDD connected to ... 1
 - POWER pin LED3 VDD connected to ... 1
 - POWER pin LED4 VDD connected to ... 1
 - Part FRAME1 has no value 1

Les led NeoPixel doivent bien être connectés au VCC de la pile et donc de l'attiny.

2.3 Calcul du coût de la carte

Désignation	Site	Référence	Unité de vente	1 exemplaire			100 exemplaires			Total
				Prix unitaire	Quantité	Total	Unité de vente	Prix unitaire	Quantité	
Condensateur C-EUC1206	Farnell	12065C104K4T2A	10	0,10 €	1	0,99 €	10	0,08 €	100	8,07 €
Support pile	Farnell	MP000362	50	26,08 €	1	26,08 €	50	20,87 €	2	41,74 €
MicroContrôleur	Farnell	ATTINY1634R-SU	1	1,76 €	1	1,76 €	1	1,50 €	100	150,00 €
Connecteur ISP	Farnell	MC-254-06-00-ST-DIP	10	0,16 €	10	1,63 €	10	0,16 €	100	16,30 €
Led Neopixel	GoTronic	16909	10	5,80 €	1	5,80 €	10	5,80 €	40	232,00 €
Mosfet canal N	Farnell	BS170F	10	0,55 €	10	5,46 €	10	0,34 €	100	34,40 €
Résistance	Farnell	LTR18EZPJ103	10	0,12 €	10	1,21 €	10	0,05 €	100	4,81 €
Buzzer 5v 2,3kHz 50ma	GoTronic	5459	1	1,80 €	1	1,80 €	1	1,80 €	100	180,00 €
Bouton poussoir	Farnell	FSM2JSMAA	10	0,13 €	10	1,29 €	10	0,13 €	400	51,60 €
Interrupteur	Farnell	EVQP7A01P	10	0,18 €	10	1,80 €	10	0,18 €	100	18,00 €
				TOTAL 1 Simon			TOTAL 100 Simon			736,92 €
				Prix rapporté à 1 Simon						7,37 €

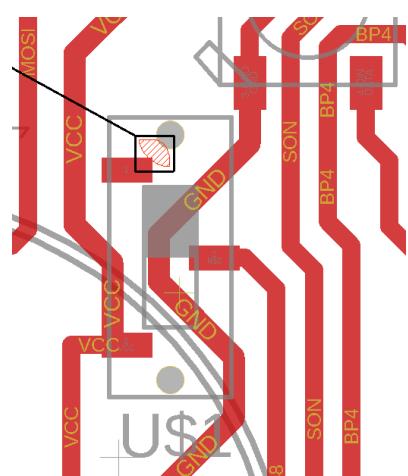
Il est intéressant de remarquer que les quantités minimum d'achat font exploser le coût de revient d'un unique Simon.

3 Routage de la carte

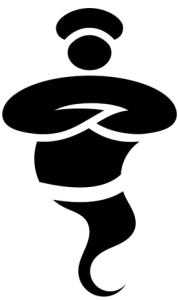
3.1 DRC

Type

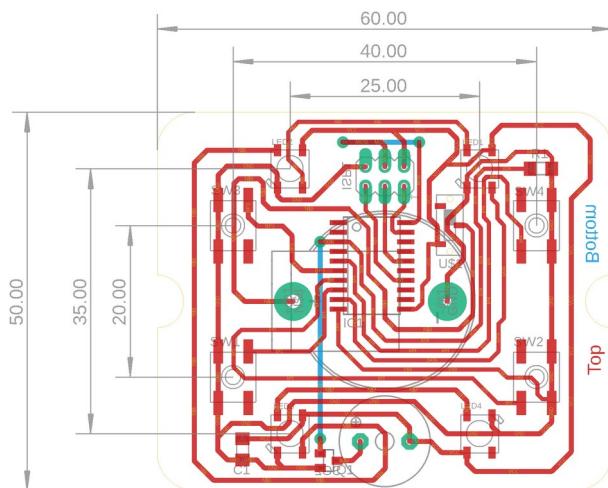
- Errors (0)
- Approved (6)
 - Dimension (4)
 - Dimension 1
 - Dimension 1
 - Dimension 1
 - Dimension 1
 - No Vector Font (2)
 - No Vector Font 1
 - No Vector Font 16



Les erreurs de police d'écriture habituels sont approuvées. Celles indiquant une proximité entre les pad et les trous de l'interrupteur aussi, c'est une erreur intrinsèque à l'empreinte.

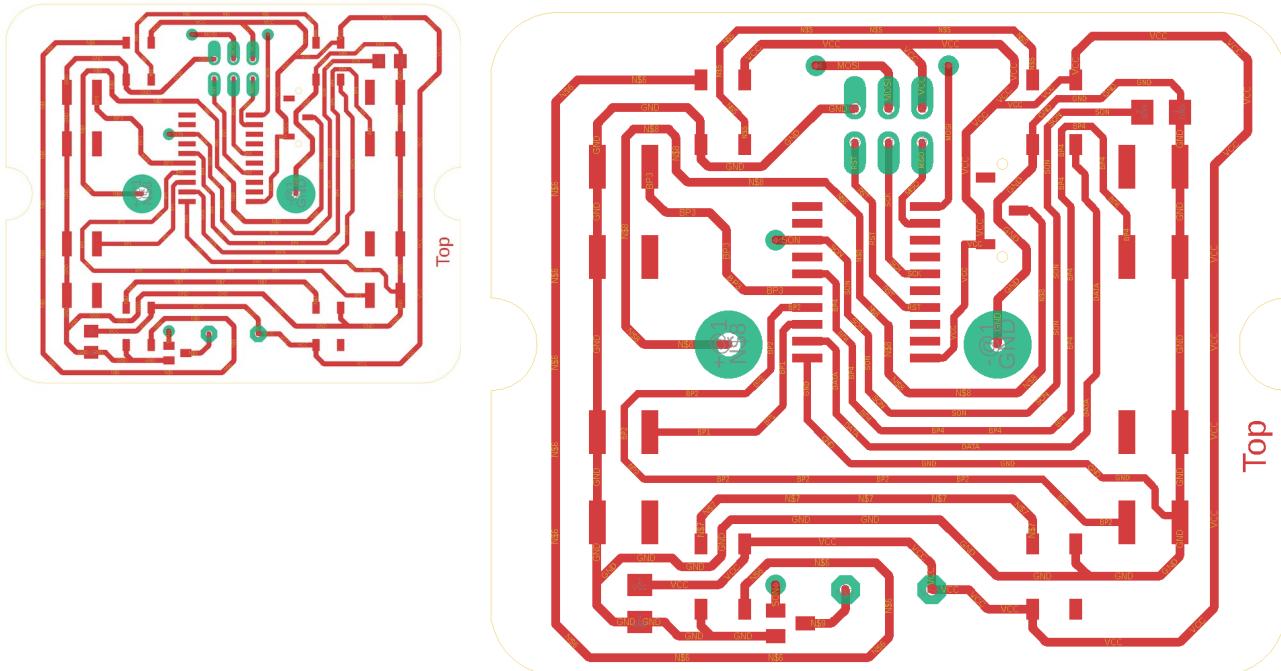


3.2 Remarques

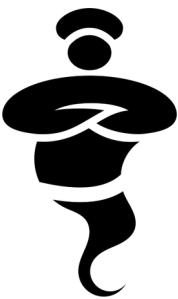


Pour maximiser l'ergonomie, j'ai déterminé l'écartement des interrupteurs à l'aide d'un réglet et d'une feuille de papier simulant le boîtier. Les autres composants sont centrés ou placés stratégiquement pour faciliter la conception du boîtier.

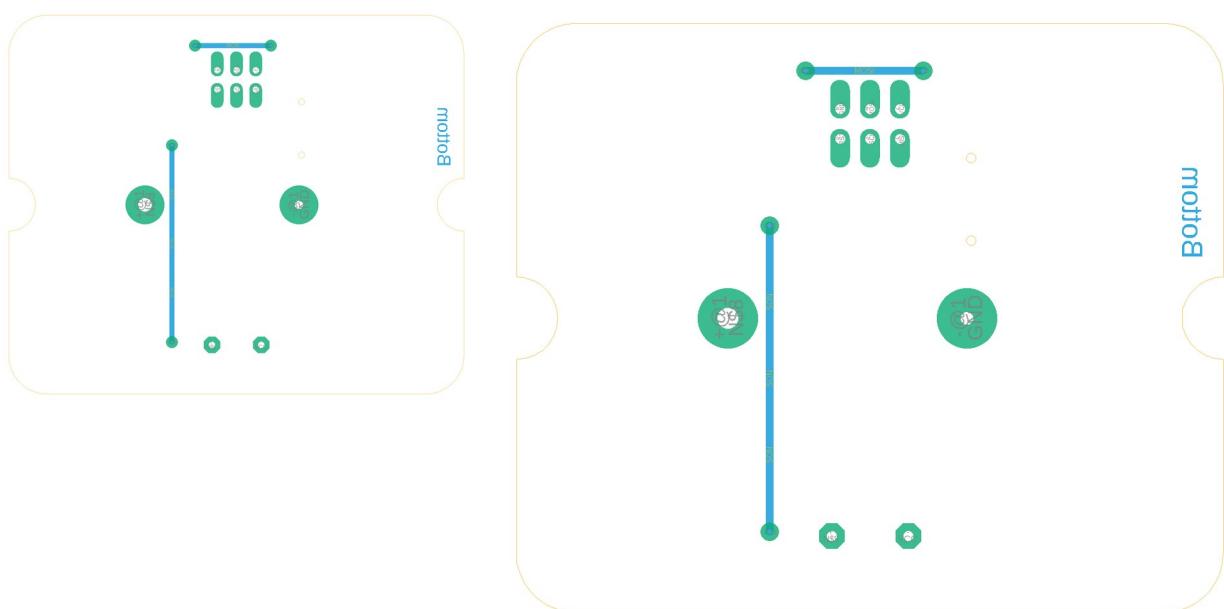
3.3 Pistes cotés top



L'espacement minimum de 15 mils est respecté, la plus part des pistes sont en 20 mils. Seules quelques unes sont en 24 mils, celle transportant un courant plus important en priorité. (Pour une CR2032, même en 20 mils c'est sûrement large)

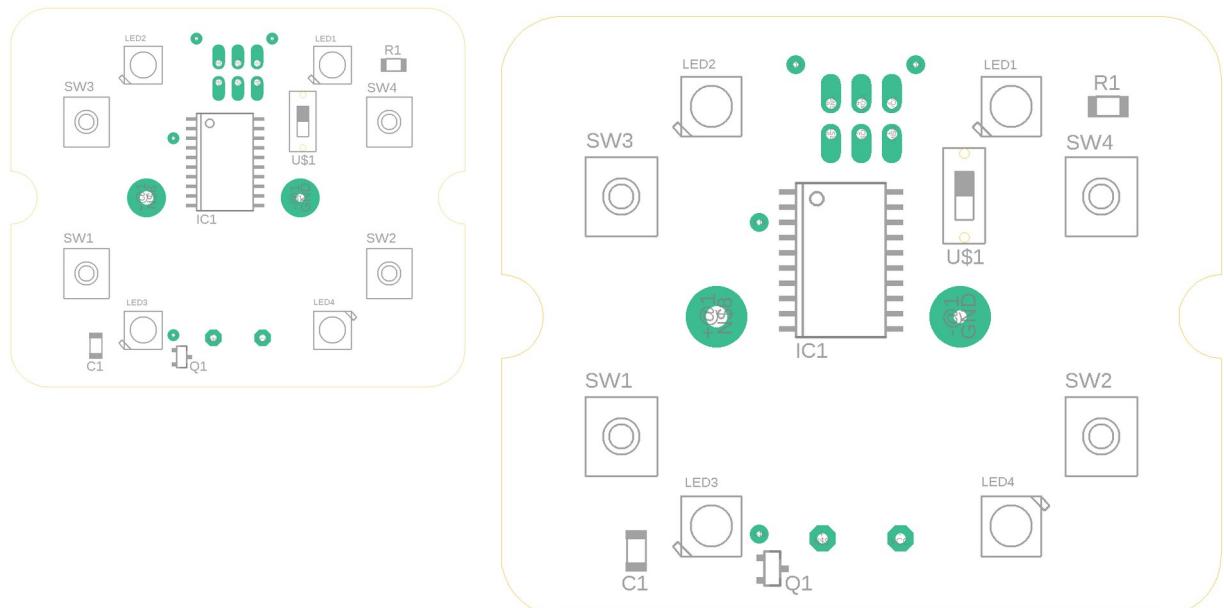


3.4 Pistes cotés bottom



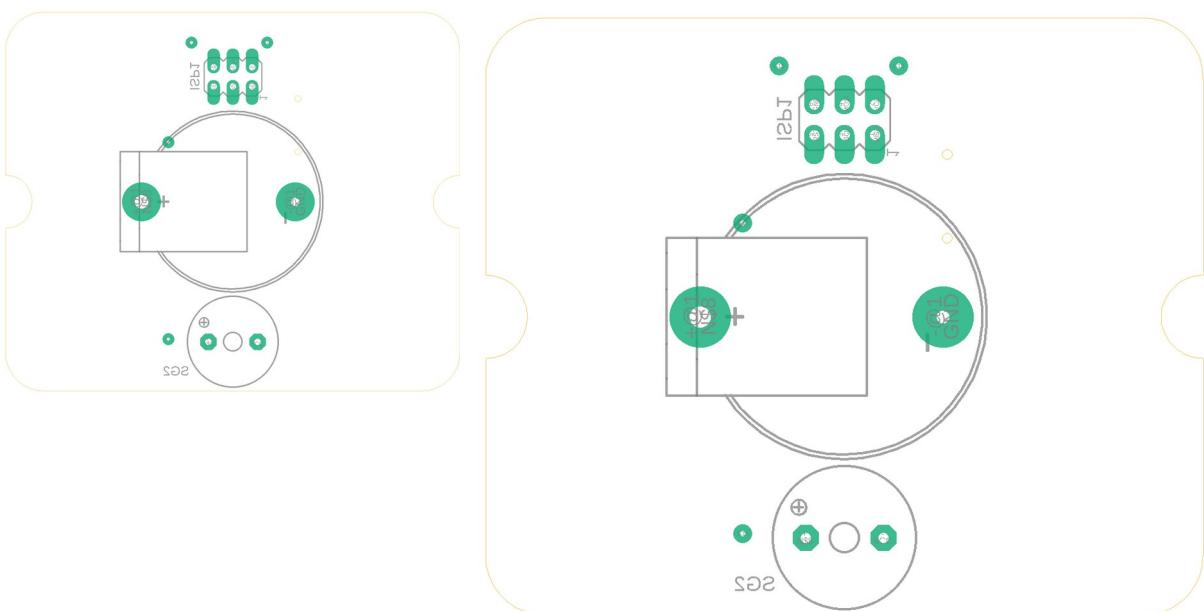
Deux pont, un qui passe sous le boîtier à pile et un autre à côté du connecteur isp.

3.5 Composants cotés top



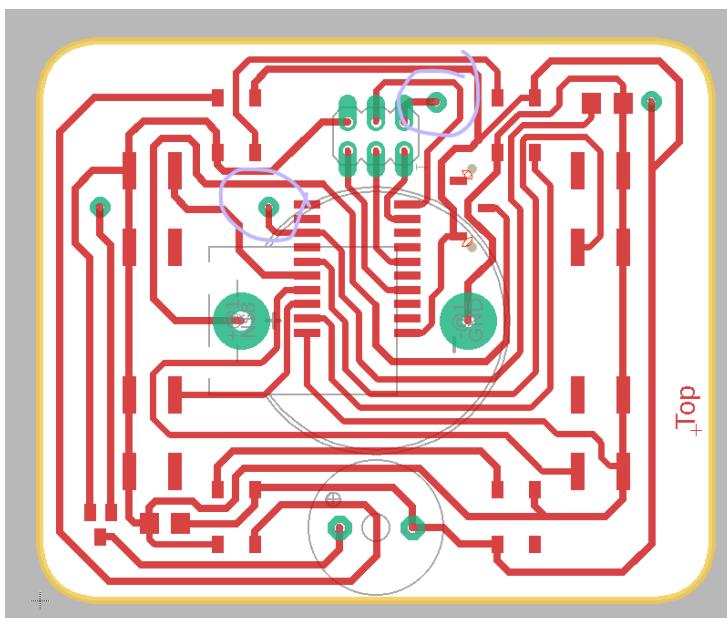
J'ai fait attention au placement du nom des composants, pour faciliter la lecture lors de la fabrication. J'ai aussi peux de composants sur les bords droits et gauches, idéal pour supporter le circuit par dessus ou le fixer grâce au couvercle par dessus.

3.6 Composants cotés bottom



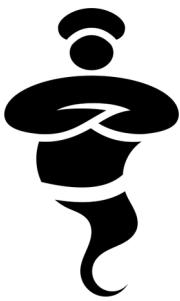
Je pourrais légèrement décaler le via qui empiète sur le boîtier à pile.

3.7 Correction



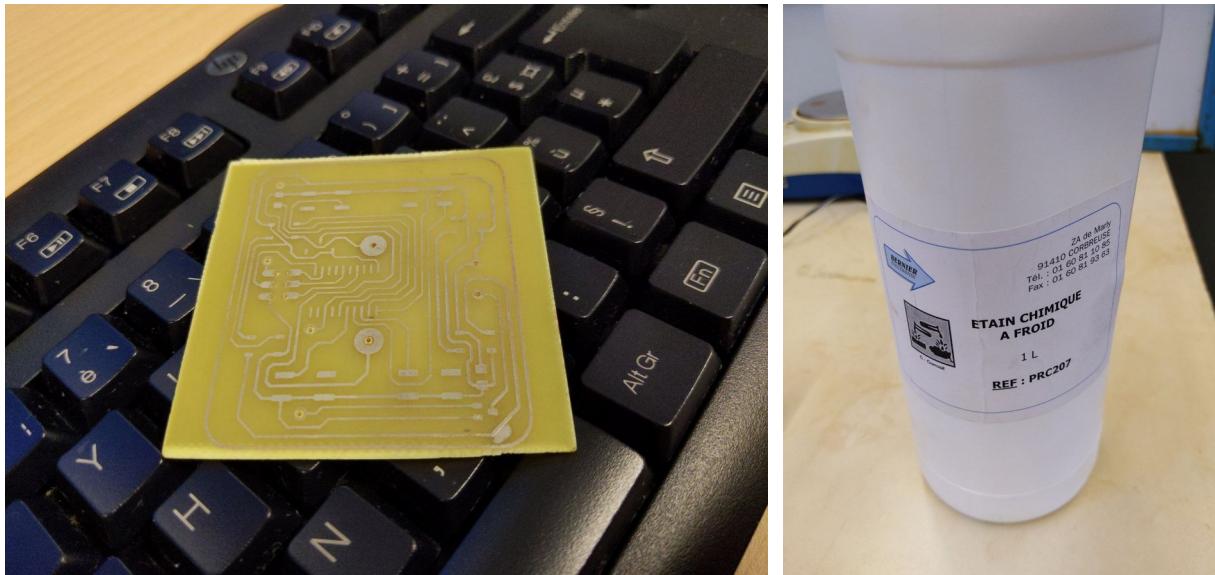
Pour faciliter la soudure des ponts, le via de gauche n'est plus sous le boîtier à pile et le via de droite est plus espacé du connecteur ISP.

Aussi, les encoches latéral de fixation ont été retirés car elles complexifiaient la découpe précise de la carte sans apporté de réel bénéfice quand à l'immobilisation du circuit.



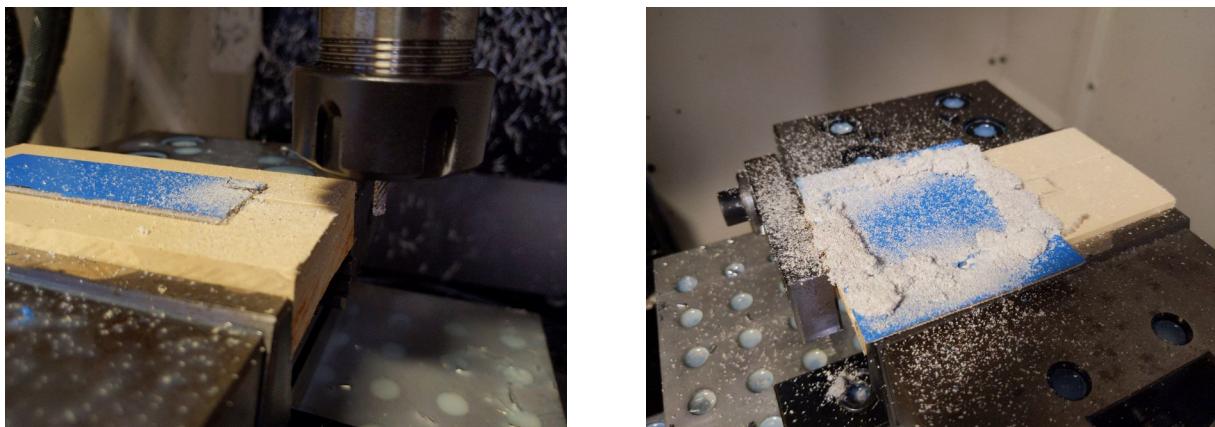
3.8 Fabrication

Une première tentative



La carte fût laissé trop longtemps dans le révélateur, 5 pistes furent sectionnées et nombre d'entre-elle étaient à un cheveu de l'être. Avec Monsieur Volpi un bain d'étain chimique à froid a été réaliser pour protéger le cuivre contre la corrosion, un succès mitigé.

Préparation de la deuxième carte

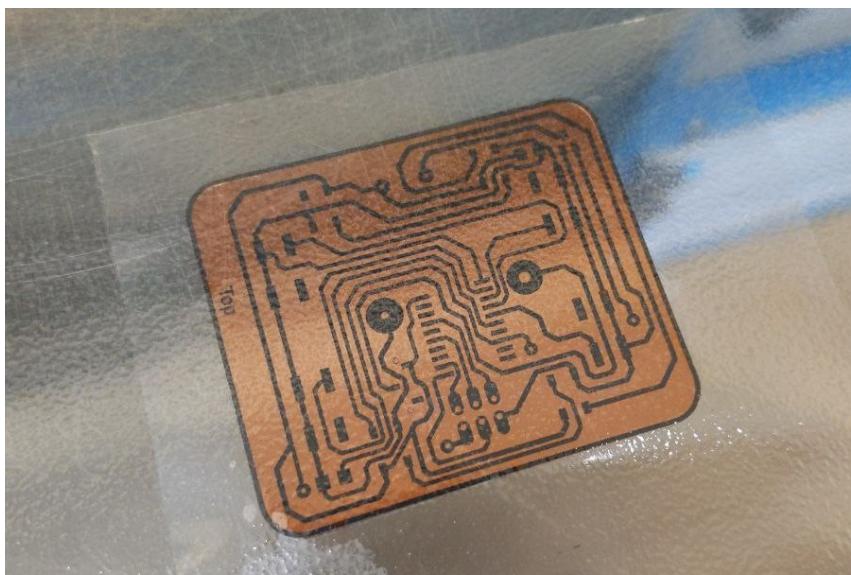


A l'aide d'une fraise spécial composite et de mon camarade, nous avons expérimenter la découper et le perçage des cartes à la fraiseuse. En effet, nos essais démontres qu'une fraise traditionnel créer beaucoup plus de bavure.



Les circuits sont fixés à l'aide de double face sur un bloc de PVC usiné spécialement pour l'occasion. (Les rainures aides à décoller le circuit) La découpe est propre, surtout après un coup de cutter. C'est pourquoi quelques circuits ont également été découpés pour mes camarades.

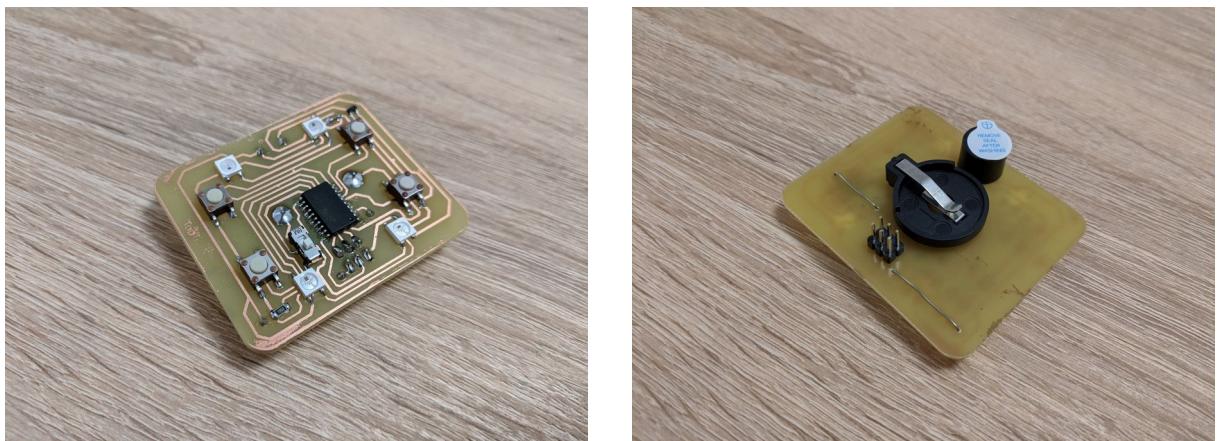
Fabrication de la deuxième carte



La gravure de la seconde carte fût couronné de succès. Le révélateur était lent, il a permis de conserver des pistes bien épiassent et parfaitement séparés.



N'ayant pas réussi à avoir un résultat satisfaisant et régulier avec la station à air chaud j'ai préféré souder la carte au fer. Par habitude serte, mais aussi car mes camarades rencontraient des difficultés avec le four à refusion.



Aucune piste n'a été sectionnée. Après vérification sous la loupe binoculaire et contrairement au premier circuit, les pistes ne sont pas criblés de trous.

Cependant, le bord de la carte n'est pas parfait, il faudra penser à écarter les pistes des bords la prochaine fois.

3.9 Boîtier



4 Concepts différents de boîtier ont été modélisé et imprimés. La première version était difficile à imprimer, les deux pièces étaient simplement alignés grâce aux vis et en plus ce n'était vraiment pas ergonomique. De plus, les boutons du couvercles étaient bien trop flexible.



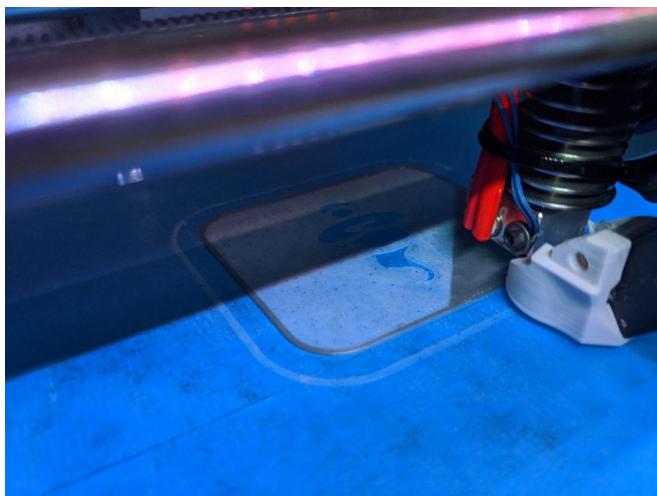
La version final est ergonomique, original et robuste. L'interrupteur marche arrêt dépasse juste ce qu'il faut pour ne pas se déclencher dans la poche.



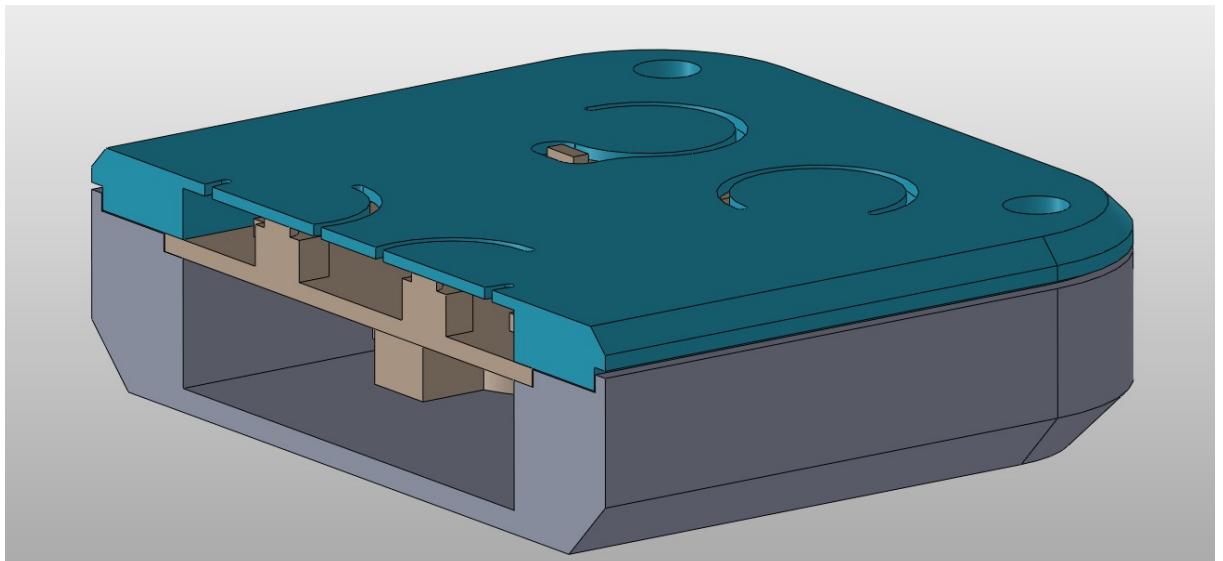
J'aime beaucoup le look de la deuxième itération mais l'arrêté du dessus était bien trop saillante.



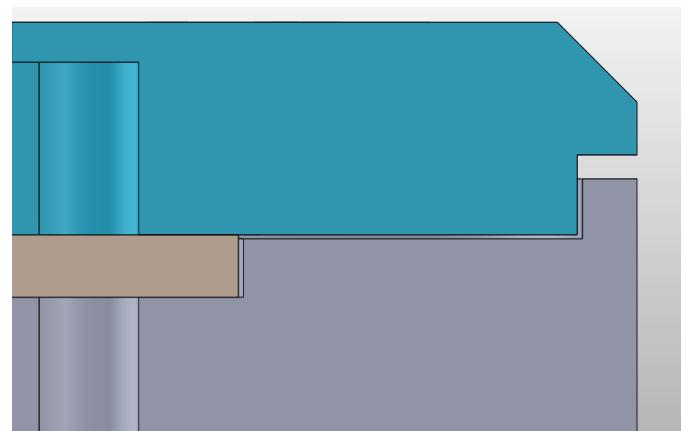
Le couvercle est suffisamment fin pour laisser passer la lumière des neopixels, il a été imprimé en blanc pour la version final.



En utilisant deux filament et en suivant une méthode spécifique au logiciel de l'imprimante, il est possible de réaliser des motifs contrastés sur des surfaces planes.



En appliquant les notions vu en cours avec Mr Rigaud et Mr Astier, j'ai su ajuster les jeux et assurer des conditions de serrages. En effet, cela assure un montage ais   et précis tout en immobilisant parfaitement le circuit.

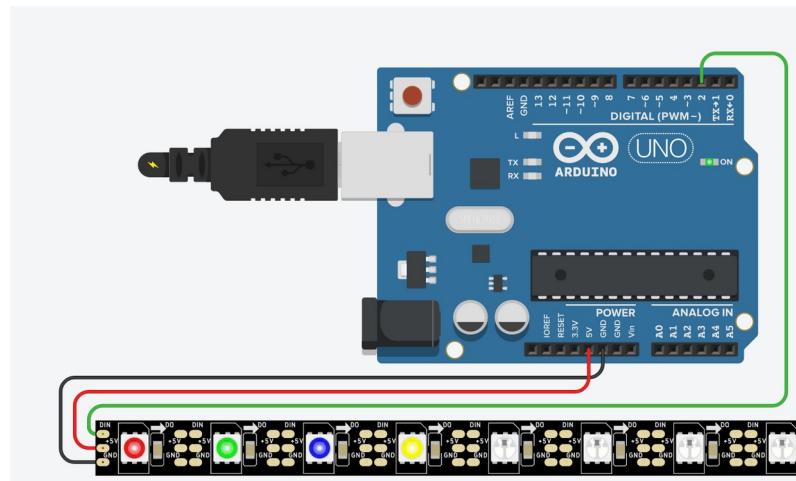


Des vis auto-foreuse sont id  ales dans le plastique.



4 Programme de validation

4.1 Les NeoPixels



Comportement de base NeoPixel: [Projet TinkerCad](#)

Ecriture d'une fonction NeoPixel : [Projet TinkerCad](#)

Comportement de base

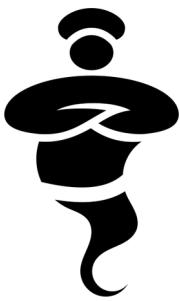
Ce programme allume la première led en rouge, la seconde en vert, la troisième en bleu et la quatrième en jaune. Il permet d'apprendre à programmer des NeoPixels et vérifie le bon fonctionnement de l'électronique.

```
// https://html-color-codes.info/ Roue de couleurs hex
#include <Adafruit_NeoPixel.h> // Importation de la bibliothèque pour gérer les NeoPixels
#define BrocheNeoPixel 2 // Broche sur laquelle les NeoPixels sont connectées
#define NombrePixel 4 // Nombre de NeoPixel

Adafruit_NeoPixel MesLed = Adafruit_NeoPixel(NombrePixel, BrocheNeoPixel, NEO_GRB + NEO_KHZ800); // Affectation des paramètres définies ci-dessus.

void setup() {
    MesLed.begin(); // Initialiser la bibliothèque
}

void loop() {
    MesLed.setPixelColor(0, MesLed.Color(150, 0, 0)); // La première led sera rouge
    MesLed.setPixelColor(1, MesLed.Color(0, 150, 0)); // La seconde led sera vert
    MesLed.setPixelColor(2, MesLed.Color(0, 0, 150)); // La troisième led sera bleu
    MesLed.setPixelColor(3, 0xFFFF00); // La quatrième led sera jaune
    MesLed.show(); // Allumer toutes les led d'un coup
}
```



Écriture d'une fonction

Ce programme utilise une fonction pour allumer les led dans la même séquence que le programme précédent. Cette fonction simplifie l'interaction avec les led.

```
// https://html-color-codes.info/ Roue de couleurs hex
#include <Adafruit_NeoPixel.h> // Importation de la bibliothèque pour gérer les NeoPixels
#define BrocheNeoPixel 2 // Broche sur laquelle les NeoPixels sont connectées
#define NombrePixel 4 // Nombre de NeoPixel

Adafruit_NeoPixel MesLed = Adafruit_NeoPixel(NombrePixel, BrocheNeoPixel, NEO_GRB + NEO_KHZ800); // Affectation des paramètres définies ci-dessus.

void setup() {
    MesLed.begin(); // Initialiser la bibliothèque
}

void allume_led(String couleur, int broche) { // Procédure prenant la couleur et le numéro d'une led en paramètre
    if (couleur == "rouge") { // Si la couleur demandé est rouge
        MesLed.setPixelColor(broche, MesLed.Color(150, 0, 0)); // régler la led spécifié sur rouge
    }
    if (couleur == "vert") { // Si la couleur demandé est vert
        MesLed.setPixelColor(broche, MesLed.Color(0, 150, 0)); // régler la led spécifié sur vert
    }
    if (couleur == "bleu") { // Si la couleur demandé est bleu
        MesLed.setPixelColor(broche, MesLed.Color(0, 0, 150)); // régler la led spécifié sur bleu
    }
    if (couleur == "jaune") { // Si la couleur demandé est jaune
        MesLed.setPixelColor(3, 0xFFFF00); // régler la led spécifié sur jaune
    }
}

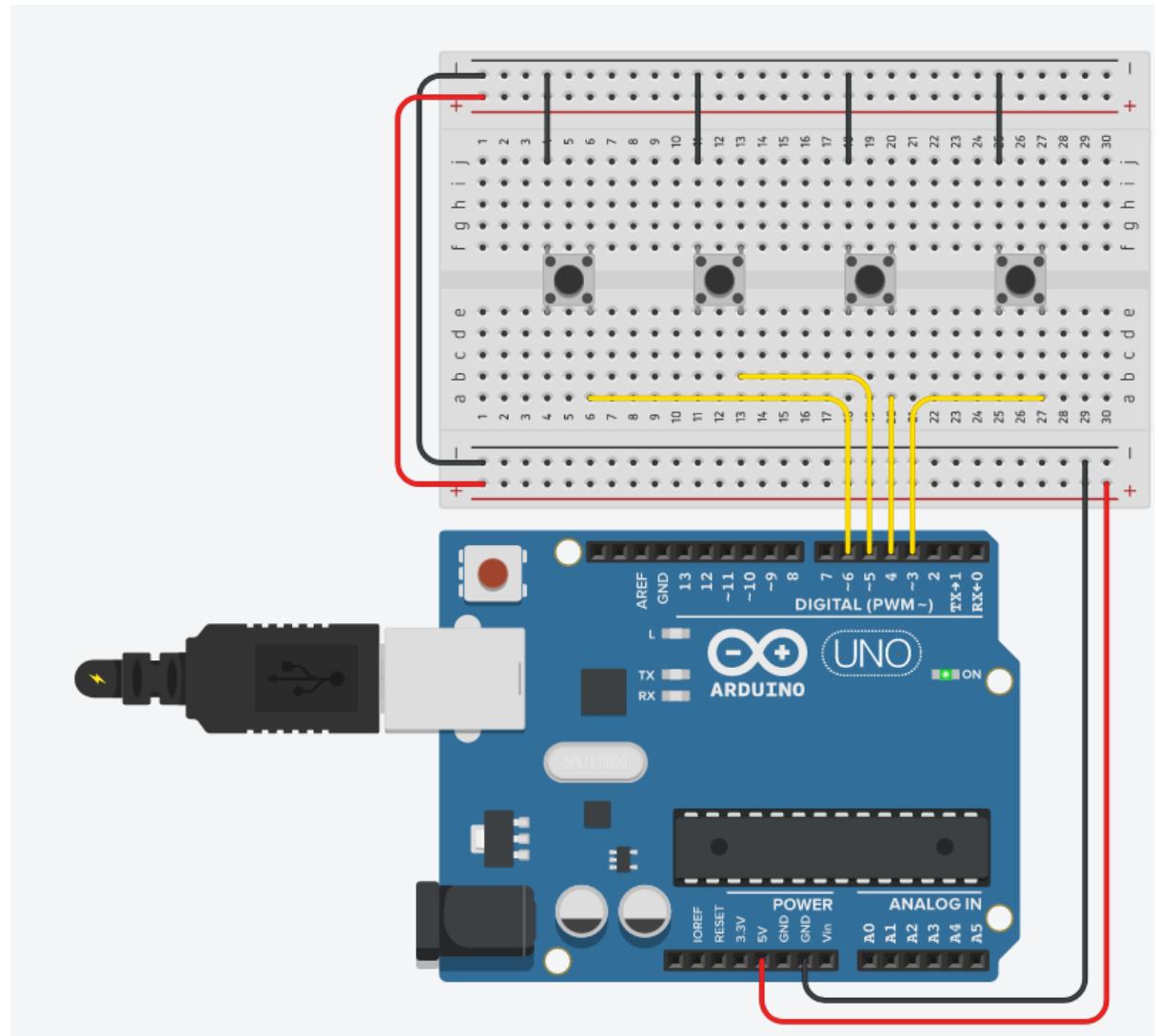
void loop() {
    allume_led("rouge", 0); // Appeler la procédure "allume_led" en lui demandant de régler la led n°0 en rouge
    allume_led("vert", 1); // Led n°1 vert
    allume_led("bleu", 2); // Led n°2 bleu
    allume_led("jaune", 3); // Led n°3 jaune
    MesLed.show(); // Allumer toutes les led d'un coup
}
```



4.2 Les boutons poussoirs

Comportement de base

Ce programme indique le numéro du bouton pressé dans le moniteur série. Il n'identifie pas la pression simultanée de plusieurs boutons.



Comportement de base Boutons : [Projet TinkerCad](#)



```
#define button1 3 // Il y a 4 boutons, le premier est nommé "button1" puis connecté à la broche n°3
#define button2 4
#define button3 5
#define button4 6
int buttonNumber = 0; // Variable contenant le numéro du bouton pressé, 0 signifiant qu'aucun bouton n'est actionné

void setup() {
    Serial.begin(9600); // Initialisation de la liaison série
    pinMode(button1, INPUT_PULLUP); // Déclaration des broches associées aux boutons en tant qu'entrée. Utilisation d'une résistance de tirage interne
    pinMode(button2, INPUT_PULLUP);
    pinMode(button3, INPUT_PULLUP);
    pinMode(button4, INPUT_PULLUP);
}

int button() {
    if (digitalRead(button1) == LOW) { // Si le premier bouton est à l'état bas c'est qu'il est pressé
        buttonNumber = 1; // Stocker le numéro du bouton pressé dans cette variable
    }
    else if (digitalRead(button2) == LOW) {
        buttonNumber = 2;
    }
    else if (digitalRead(button3) == LOW) {
        buttonNumber = 3;
    }
    else if (digitalRead(button4) == LOW) {
        buttonNumber = 4;
    }
    else { // Revient à dire "Si ni le bouton n°1, ni le n°2, ni le n°3, ni le n°4 n'est pressé c'est que qu'aucun bouton n'est pressé"
        buttonNumber = 0;
    }
    return (buttonNumber); // Lorsque la procédure est exécuté elle retournera le numéro du bouton pressé contenu dans la variable "buttonNumber"
}

void loop() {
    buttonNumber = button(); // Executer la procédure qui va lire l'état des boutons et le retourner
    Serial.println(buttonNumber); // Afficher l'état des boutons
}
```

Petit défi personnel

```
Moniteur série
1011
Etat bouton n0->1
Etat bouton n1->0
Etat bouton n2->1
Etat bouton n3->1
```

J'ai modifié le programme ci-dessus pour identifier la pression simultanée de plusieurs boutons. Les données sont stockées dans un tableau puis l'état de chacun des boutons est affiché de manière lisible dans le moniteur série : [Projet TinkerCad](#)



```
#include <Adafruit_NeoPixel.h> // Importation de la bibliothèque pour gérer les NeoPixels
#define BrocheNeoPixel 2 // Broche sur laquelle les NeoPixels sont connectées
#define NombrePixel 4 // Nombre de NeoPixel
Adafruit_NeoPixel MesLed = Adafruit_NeoPixel(NombrePixel, BrocheNeoPixel, NEO_GRB + NEO_KHZ800); // Affectation des paramètres définies ci-dessus.

#define button0 3 // Il y a 4 boutons, le premier est nommé "button1" puis connecté à la broche n°3
#define button1 4
#define button2 5
#define button3 6
int buttonNumber[4] = {1, 1, 1, 1}; // Tableau de type int comprenant l'état des 4 boutons

void setup() {
    MesLed.begin(); // Initialiser la bibliothèque
    Serial.begin(9600); // Initialisation de la liaison série
    pinMode(button0, INPUT_PULLUP); // Déclaration des broches associées aux boutons en tant qu'entrée. Utilisation d'une résistance de tirage interne
    pinMode(button1, INPUT_PULLUP);
    pinMode(button2, INPUT_PULLUP);
    pinMode(button3, INPUT_PULLUP);
}

void button() {
    buttonNumber[0] = digitalRead(button0);
    buttonNumber[1] = digitalRead(button1);
    buttonNumber[2] = digitalRead(button2);
    buttonNumber[3] = digitalRead(button3);
}

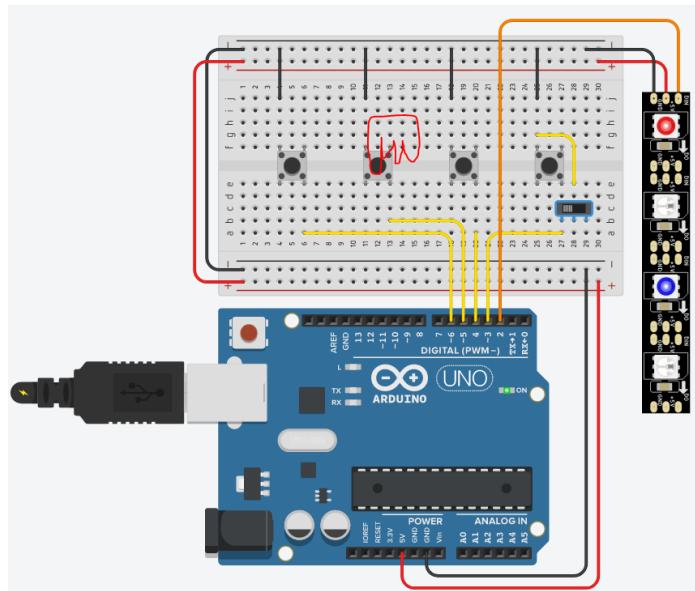
void loop() {
    button();
    for (int i = 0; i < 4; i++) {
        Serial.print(buttonNumber[i]);
        delay(100);
    }
    Serial.println();
    Serial.print("Etat bouton n0->");
    Serial.println(buttonNumber[0]);
    Serial.print("Etat bouton n1->");
    Serial.println(buttonNumber[1]);
    Serial.print("Etat bouton n2->");
    Serial.println(buttonNumber[2]);
    Serial.print("Etat bouton n3->");
    Serial.println(buttonNumber[3]);
    Serial.println();
    Serial.println();
}
```



Combinaison des NeoPixels et des boutons poussoirs

Il est désormais possible d'allumer une ou plusieurs del en appuyant sur le ou les boutons correspondants.

Pour simuler une pression simultané avec un seul curseur de souris, il a fallut ajouter un bouton à glissière (donc non momentané).



*Combinaison NeoPixel Bouton Poussoirs:
Projet TinkerCad*

```
#include <Adafruit_NeoPixel.h> // Importation de la bibliothèque pour gérer les NeoPixels
#define BrocheNeoPixel 2 // Broche sur laquelle les NeoPixels sont connectées
#define NombrePixel 4 // Nombre de NeoPixel
Adafruit_NeoPixel MesLed = Adafruit_NeoPixel(NombrePixel, BrocheNeoPixel, NEO_GRB + NEO_KHZ800); // Affectation des paramètres définies ci-dessus.

#define button0 3 // Il y a 4 boutons, le premier est nommé "button1" puis connecté à la broche n°3
#define button1 4
#define button2 5
#define button3 6
int buttonNumber[4] = {1, 1, 1, 1}; // Tableau de type int comprenant l'état des 4 boutons

void setup() {
    MesLed.begin(); // Initialiser la bibliothèque
    Serial.begin(9600); // Initialisation de la liaison série
    pinMode(button0, INPUT_PULLUP); // Déclaration des broches associées aux boutons en tant qu'entrée. Utilisation d'une résistance de tirage interne
    pinMode(button1, INPUT_PULLUP);
    pinMode(button2, INPUT_PULLUP);
    pinMode(button3, INPUT_PULLUP);
}
```



```
void button() {
    buttonNumber[0] = digitalRead(button0); // La case n°0 du tableau contient l'état du bouton
    n°0 en cet instant
    buttonNumber[1] = digitalRead(button1);
    buttonNumber[2] = digitalRead(button2);
    buttonNumber[3] = digitalRead(button3);
}

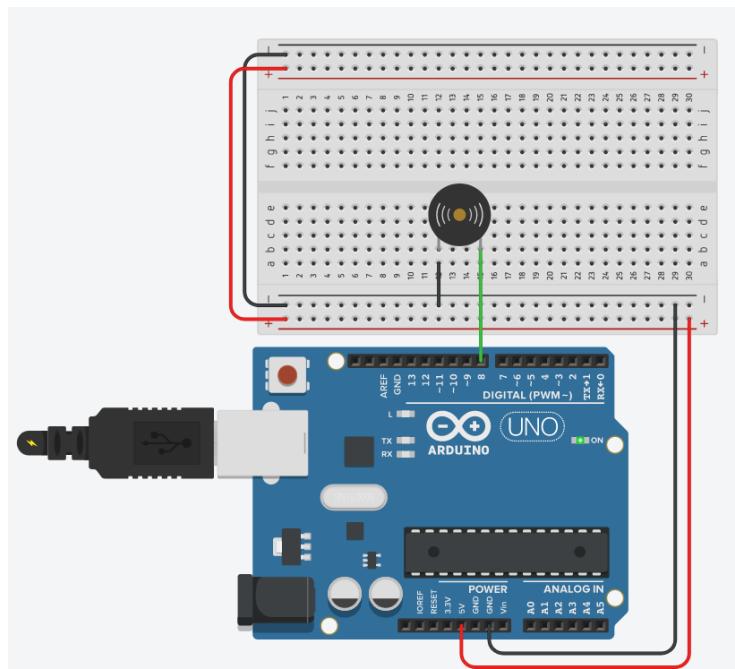
void led() {
    if (buttonNumber[0] == 0) { // Si le bouton n°0 est à l'état bas c'est qu'il est pressé
        MesLed.setPixelColor(0, MesLed.Color(150, 0, 0)); // Alors, la led n°0 sera verte
    }
    else { // Si le bouton n°0 n'est pas pressé
        MesLed.setPixelColor(0, MesLed.Color(0, 0, 0)); // Alors, éteindre la led n°0
    }
    if (buttonNumber[1] == 0) {
        MesLed.setPixelColor(1, MesLed.Color(0, 150, 0));
    }
    else {
        MesLed.setPixelColor(1, MesLed.Color(0, 0, 0));
    }
    if (buttonNumber[2] == 0) {
        MesLed.setPixelColor(2, MesLed.Color(0, 0, 150));
    }
    else {
        MesLed.setPixelColor(2, MesLed.Color(0, 0, 0));
    }
    if (buttonNumber[3] == 0) {
        MesLed.setPixelColor(3, 0xFFFF00);
    }
    else {
        MesLed.setPixelColor(3, MesLed.Color(0, 0, 0));
    }
}

void loop() {
    button(); // Executer la procédure pour lire l'état des boutons
    led(); // Executer la procédure pour déterminer les leds à allumer
    MesLed.show(); // Allumer les led ou plus précisement rafraîchir l'état des led
}
```



4.3 Le Buzzer

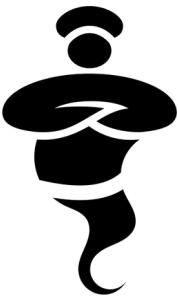
Comportement de base



Buzzer, Comportement de base: [Projet TinkerCad](#)

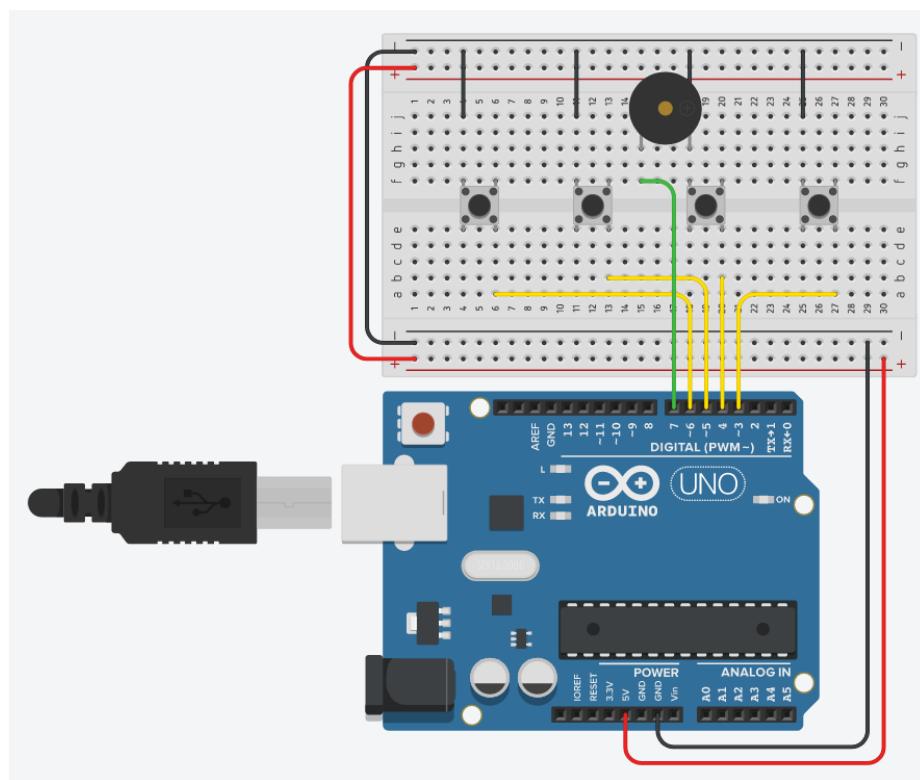
Ce programme joue un Là une octave après l'autre. Il est intéressant de noté que l'exécution du programme est fortement ralenti lors de la génération d'un sons, d'où les grésillements.

```
#define buzzerPin 8 // Le buzzer est connecté à la broche n°8
void setup() {
    pinMode(buzzerPin, OUTPUT); // La broche utilisé par le buzzer est configuré en tant que sortie
}
void loop() {
    tone(buzzerPin, 110); // Jouer un Là en première octave, 110Hz
    delay(500); // Pendant une demi seconde
    noTone(buzzerPin); // Eteindre le buzzer
    delay(1500); // Pendant 1,5 sec
    tone(buzzerPin, 220); // Jouer un Là en seconde octave, 220Hz
    delay(500);
    noTone(buzzerPin);
    delay(1500);
    tone(buzzerPin, 440); // Jouer un Là en troisième octave, 440Hz
    delay(500);
    noTone(buzzerPin);
    delay(1500);
    tone(buzzerPin, 880); // Jouer un Là en quatrième octave, 880Hz
    delay(500);
    noTone(buzzerPin);
    delay(1500);
}
```



Combinaison boutons pousoirs et NeoPixels

Lorsqu'un bouton est pressé, le buzzer joue le son associé.



Simon_P6.1_Combinaison Buzzer Boutons: [Projet TinkerCad](#)

```
#define button1 3 // Il y a 4 boutons, le premier est nommé "button1" puis connecté à la
broche n°3
#define button2 4
#define button3 5
#define button4 6
int buttonNumber = 0; // Variable contenant le numéro du bouton pressé, 0 signifiant
qu'aucun bouton n'est actionné
#define buzzerPin 7 // Le buzzer est connecté à la broche n°8

void setup() {
    pinMode(button1, INPUT_PULLUP); // Déclaration des broches associées aux boutons en
tant qu'entrée. Utilisation d'une résistance de tirage interne
    pinMode(button2, INPUT_PULLUP);
    pinMode(button3, INPUT_PULLUP);
    pinMode(button4, INPUT_PULLUP);
    pinMode(buzzerPin, OUTPUT); // La broche utilisé par le buzzer est configuré en tant que
sortie
}
```



```
int button() {
    if (digitalRead(button1) == LOW) { // Si le premier bouton est à l'état bas c'est qu'il est pressé
        buttonNumber = 1; // Stocker le numéro du bouton pressé dans cette variable
    }
    else if (digitalRead(button2) == LOW) {
        buttonNumber = 2;
    }
    else if (digitalRead(button3) == LOW) {
        buttonNumber = 3;
    }
    else if (digitalRead(button4) == LOW) {
        buttonNumber = 4;
    }
    else { // Revient à dire "Si ni le bouton n°1, ni le n°2, ni le n°3, ni le n°4 n'est pressé c'est que qu'aucun bouton n'est pressé"
        buttonNumber = 0;
    }
    return (buttonNumber); // Lorsque la procédure est executé elle retournera le numéro du bouton pressé contenu dans la variable "buttonNumber"
}

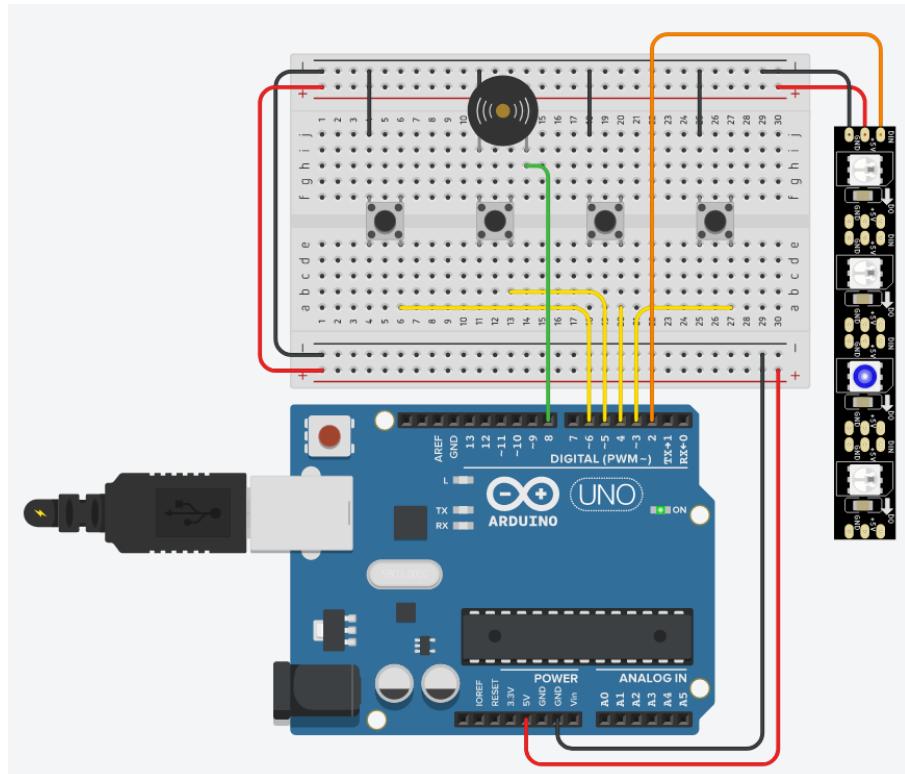
void buzzer() {
    switch (buttonNumber) { // On s'intéresse à la valeur stocké dans la variable buttonNumber
        case 1: // Si elle vaut 1 c'est que le premier bouton est pressé
            tone(buzzerPin, 110); // Jouer un Là à 110Hz
            break;
        case 2:
            tone(buzzerPin, 220);
            break;
        case 3:
            tone(buzzerPin, 440);
            break;
        case 4:
            tone(buzzerPin, 880);
            break;
        default: // Si elle ne vaut ni 1, ni 2, ni 3, ni 4, c'est qu'elle vaut 0 donc aucun bouton n'est pressé
            noTone(buzzerPin); // Eteindre le buzzer
            break;
    }
}

void loop() {
    buttonNumber = button(); // Executer la fonction qui va lire l'état des boutons et l'enregistrer dans la variable buttonNumber
    buzzer(); // Executer la procédure qui va jouer un son en fonction du bouton pressé
}
```



4.4 Presque un Simon !

Ce programme synthétise toutes les notions apprise précédemment. Lorsqu'un bouton est pressé il allume la led et le son associé.



Validation Final: [Projet TinkerCad](#)

```
#include <Adafruit_NeoPixel.h> // Importation de la bibliothèque pour gérer les NeoPixels
#define BrocheNeoPixel 2 // Broche sur laquelle les NeoPixels sont connectées
#define NombrePixel 4 // Nombre de NeoPixel
Adafruit_NeoPixel MesLed = Adafruit_NeoPixel(NombrePixel, BrocheNeoPixel, NEO_GRB + NEO_KHZ800); // Affectation des paramètres définies ci-dessus.

#define button1 3 // Il y a 4 boutons, le premier est nommé "button1" puis connecté à la
// broche n°3
#define button2 4
#define button3 5
#define button4 6
int buttonNumber = 0; // Variable contenant le numéro du bouton pressé, 0 signifiant qu'aucun
// bouton n'est actionné
#define buzzerPin 8 // Le buzzer est connecté à la broche n°8
```



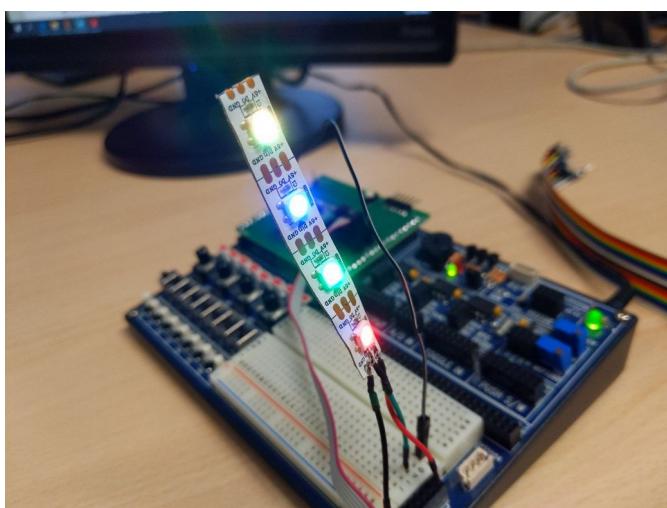
```
void setup() {
    MesLed.begin(); // Initialiser la bibliothèque
    pinMode(button1, INPUT_PULLUP); // Déclaration des broches associées aux boutons en tant qu'entré. Utilisation d'une résistance de tirage interne
    pinMode(button2, INPUT_PULLUP);
    pinMode(button3, INPUT_PULLUP);
    pinMode(button4, INPUT_PULLUP);
    pinMode(buzzerPin, OUTPUT); // La broche utilisé par le buzzer est configuré en tant que sortie
}

int button() {
    if (digitalRead(button1) == LOW) { // Si le premier bouton est à l'état bas c'est qu'il est pressé
        buttonNumber = 1; // Stocker le numéro du bouton pressé dans cette variable
    }
    else if (digitalRead(button2) == LOW) {
        buttonNumber = 2;
    }
    else if (digitalRead(button3) == LOW) {
        buttonNumber = 3;
    }
    else if (digitalRead(button4) == LOW) {
        buttonNumber = 4;
    }
    else { // Revient à dire "Si ni le bouton n°1, ni le n°2, ni le n°3, ni le n°4 n'est pressé c'est que qu'aucun bouton n'est pressé"
        buttonNumber = 0;
    }
    return (buttonNumber); // Lorsque la procédure est executé elle retournera le numéro du bouton pressé contenu dans la variable "buttonNumber"
}

void buzzer() {
    switch (buttonNumber) { // On s'intéresse à la valeur stocké dans la variable buttonNumber
        case 1: // Si elle vaut 1 c'est que le premier bouton est pressé
            tone(buzzerPin, 110); // Jouer un Là à 110Hz
            break;
        case 2:
            tone(buzzerPin, 220);
            break;
        case 3:
            tone(buzzerPin, 440);
            break;
        case 4:
            tone(buzzerPin, 880);
            break;
        default: // Si elle ne vaut ni 1, ni 2, ni 3, ni 4, c'est qu'elle vaut 0 donc aucun bouton n'est pressé
            noTone(buzzerPin); // Eteindre le buzzer
            break;
    }
}
```



```
void led() {  
    switch (buttonNumber) { // On s'intéresse à la valeur stocké dans la variable buttonNumber  
        case 1: // Si elle vaut 1 c'est que le premier bouton est pressé  
            MesLed.setPixelColor(0, MesLed.Color(150, 0, 0)); // Alors, la led n°0 sera verte  
            break;  
        case 2:  
            MesLed.setPixelColor(1, MesLed.Color(0, 150, 0));  
            break;  
        case 3:  
            MesLed.setPixelColor(2, MesLed.Color(0, 0, 150));  
            break;  
        case 4:  
            MesLed.setPixelColor(3, 0xFFFF00);  
            break;  
        default: // Si elle ne vaut ni 1, ni 2, ni 3, ni 4, c'est qu'elle vaut 0 donc aucun bouton n'est  
        pressé  
            MesLed.setPixelColor(0, MesLed.Color(0, 0, 0)); // Alors les led doivent-être éteintes  
            MesLed.setPixelColor(1, MesLed.Color(0, 0, 0));  
            MesLed.setPixelColor(2, MesLed.Color(0, 0, 0));  
            MesLed.setPixelColor(3, MesLed.Color(0, 0, 0));  
            break;  
    }  
}  
  
void loop() {  
    buttonNumber = button(); // Executer la fonction qui va lire l'état des boutons et l'enregistrer  
    dans la variable buttonNumber  
    buzzer(); // Executer la procédure qui va jouer un son en fonction du bouton pressé  
    led(); // Executer la procédure pour déterminer les leds à allumer  
    MesLed.show(); // Allumer les led ou plus précisément rafraîchir l'état des led  
}
```



Attinyyyy !



5 Algorithme général

5.1 Générer une séquence de chiffre vraiment aléatoire

```
Moniteur série
4
6
4
9
0
6
9
9
```

Suite de chiffre vraiment aléatoire:

[Projet TinkerCad](#)

Ce programme génère des groupes de 4 chiffres réellement aléatoires grâce à différentes variantes de randomSeed(); En effet, il faut générer une "graine" différente régulièrement pour que l'algorithme random(); ne génère pas tout le temps les mêmes chiffres.

```
int number = 0;
// void(* resetFunc) (void) = 0; // Lorsqu'elle est appelé, cette fonction redémarre l'arduino
// (reset)
/* ATTENTION la fonction de redémarrage ne marche pas bien sur tinkercad !
 * Elle remplace le premier chiffre par un j...
 */
void setup() {
    Serial.begin(9600);

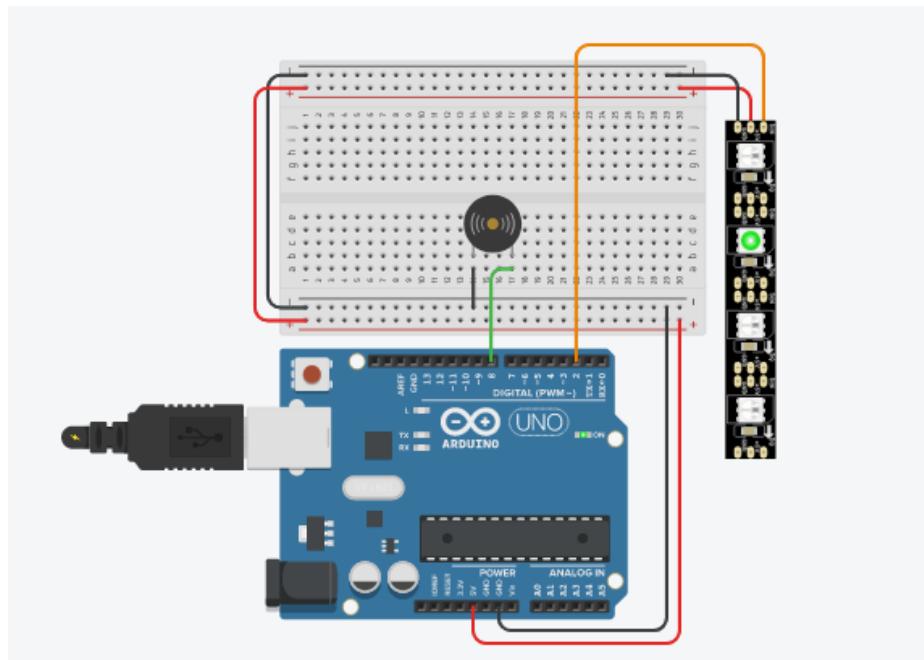
    randomSeed(analogRead(0)); // ok tinkercad, ok en physique
    //randomSeed(micros()); // Ca ne marche pas dans tinkercad, ok en physique

    for (int i = 0; i < 4; i++) { // Générer 4 chiffres aléatoires compris entre 0 et 9.
        number = random(10);
        Serial.println(number); // Les afficher
    }
    Serial.println(); // Laisser un espace
    delay(2000); // Attendre un peu
    // resetFunc(); // Puis redémarrer l'arduino à fin de générer le prochain groupe de chiffre
}

void loop() {
```



5.2 Afficher la séquence de chiffre aléatoire



Les NeoPixel et le Buzzer montre la séquence de chiffre : [Projet TinkerCad](#)

Ce programme créer une suite de chiffre traduite en son et lumière. A chaque "tour" la suite s'allonge d'un chiffre, jusqu'à une longueur de 5 caractères avant de recommencer. Sur un vrai Simon, la longueur de la suite serait de 31 caractère et la partie serait gagné une fois arrivé à cette longueur max.

```
#include <Adafruit_NeoPixel.h> // Importation de la
bibliotheque pour gerer les NeoPixels
#define NeoPixelPin 2 // Broche sur laquelle
les NeoPixels sont connectees
#define PixelCount 4 // Nombre de NeoPixel
Adafruit_NeoPixel MyLed = Adafruit_NeoPixel(PixelCount, NeoPixelPin, NEO_GRB +
NEO_KHZ800); // Affectation des parametres definies ci-dessus.
#define buzzerPin 8 // Le buzzer est connecte
à la broche n°8
int SimonSequence[31];

void setup()
{
  Serial.begin(9600); // Initialisation d'une liaison serie pour du deboggage
  MyLed.begin(); // Initialiser la bibliotheque
}
```



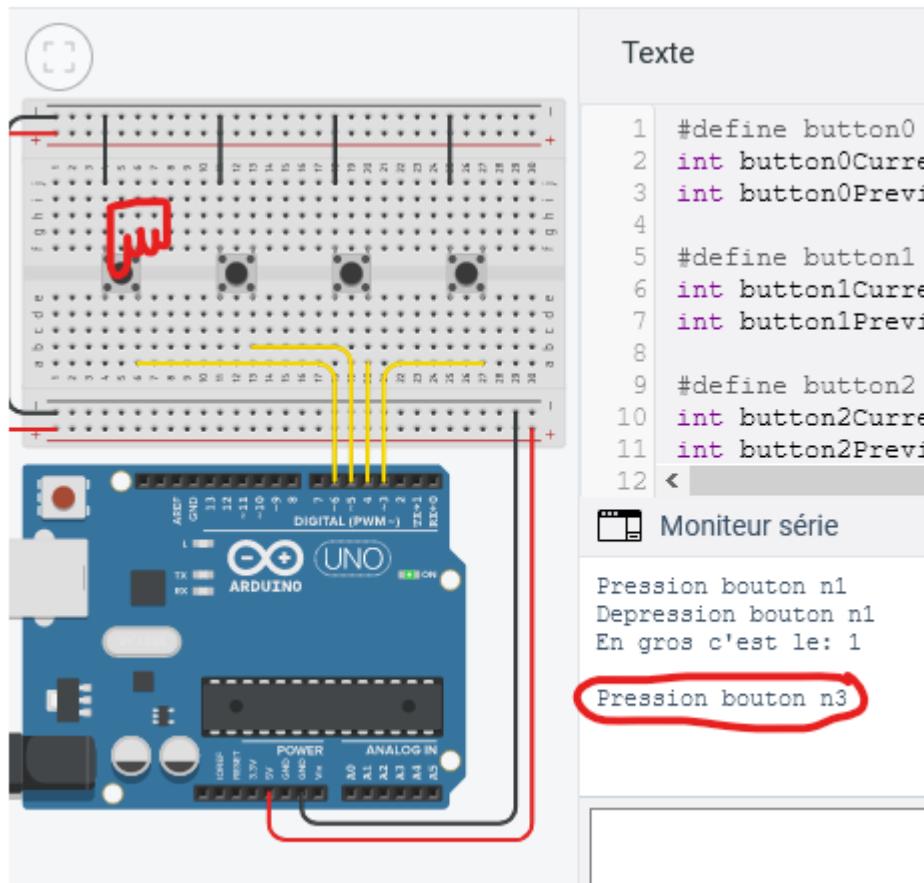
```
void OnLed(int ledNumber)
{ // Procedure prenant le numero d'une led en parametre et allumant la led correspondante
switch (ledNumber)
{
case 0:
    MyLed.setPixelColor(ledNumber, MyLed.Color(150, 0, 0)); // Si la led n°0 est demande, elle est configure en rouge
    tone(buzzerPin, 252);
    break;
case 1:
    MyLed.setPixelColor(ledNumber, MyLed.Color(0, 150, 0)); // Si la led n°1 est demande, elle est configure en vert
    tone(buzzerPin, 209);
    break;
case 2:
    MyLed.setPixelColor(ledNumber, MyLed.Color(0, 0, 150)); // Si la led n°2 est demande, elle est configure en bleu
    tone(buzzerPin, 415);
    break;
case 3:
    MyLed.setPixelColor(3, 0xFFFF00); // Si la led n°3 est demande, elle est configure en jaune
    tone(buzzerPin, 310);
    break;
}
MyLed.show(); // Allumer la led configure
delay(420); // Pendant 420ms
noTone(buzzerPin);
MyLed.clear(); // Configurer toutes les led en "noir" (Equivalent de: MyLed.setPixelColor(broche, MyLed.Color(0, 0, 0)));
MyLed.show(); // Eteindre toutes les led
delay(50); // Pendant 50ms
}

void SimonShow(int SequenceLenght)
{
int showed = 0;
while (showed < SequenceLenght)
{
    OnLed(SimonSequence[showed]);
    showed++;
    Serial.println(SimonSequence[showed]);
}
}

void loop()
{
int SequenceLenght = 1;

for (int i = 0; i < 5; i++)
{
    randomSeed(analogRead(0));
    SimonSequence[i] = random(0, 4); // Generer un chiffre aleatoire et le stocker dans la case i du tableau
}
Serial.println("Sequence genere");
while (SequenceLenght <= 5)
{
    SimonShow(SequenceLenght);
    SequenceLenght++;
    delay(800); // Pause de 800ms avant de jouer la prochaine sequence
    Serial.println("Montrer");
}
Serial.println("fin");
}
```

5.3 Mieux détecter la pression des boutons



Rising edge and Falling edge detection with debouncing: [Projet TinkerCad](#)

Lors d'essais précédents, j'ai remarqué que l'unique pression d'un bouton pouvait être détecter comme 2 ou 3 pression rapprochés. Ce nouveau programme intègre donc des délais astucieusement placés pour remédier au problème.

Également, la pression prolongé d'un bouton était reçu comme de multiples pressions d'un même bouton. C'est un problème lorsque l'on souhaite analyser une séquence de pression. Ce nouveau programme y remédie en détectant séparément la « pression » et la « dépression » des boutons.

```

#define button0 3
int button0currentState = 0;
int button0previousState = 1;

#define button1 4
int button1currentState = 0;
int button1previousState = 1;

#define button2 5
int button2currentState = 0;
int button2previousState = 1;

#define button3 6
int button3currentState = 0;
int button3previousState = 1;

int buttonNumber = 0; // Variable contenant le numéro du bouton pressé, 0 signifiant qu'aucun bouton n'est actionné

```



```
int button() {
    button0currentState = digitalRead(button0); // Récupérer l'état actuel du bouton
    button1currentState = digitalRead(button1); // Récupérer l'état actuel du bouton
    button2currentState = digitalRead(button2); // Récupérer l'état actuel du bouton
    button3currentState = digitalRead(button3); // Récupérer l'état actuel du bouton

    if (button0currentState != button0PreviousState) { // Si cet état a changé
        if (button0currentState == LOW) { // Si cet état est passé de haut à bas (C'est inversé à cause du PullUp)
            Serial.println("Pression bouton n°0"); // C'est que le bouton est pressé, falling edge
        }
        else { // Si cet état est passé de bas à haut
            Serial.println("Dépression bouton n°0"); // C'est que le bouton est relâché, rising edge
            buttonNumber = 0;
        }
        delay(50); // Debounce ?
        button0PreviousState = button0currentState; // Fin de l'action, cet état fait donc désormais parti du passé
    }

    else if (button1currentState != button1PreviousState) { // Si cet état a changé
        if (button1currentState == LOW) { // Si cet état est passé de haut à bas (C'est inversé à cause du PullUp)
            Serial.println("Pression bouton n°1"); // C'est que le bouton est pressé, falling edge
        }
        else { // Si cet état est passé de bas à haut
            Serial.println("Dépression bouton n°1"); // C'est que le bouton est relâché, rising edge
            buttonNumber = 1;
        }
        delay(50); // Debounce ?
        button1PreviousState = button1currentState; // Fin de l'action, cet état fait donc désormais parti du passé
    }

    else if (button2currentState != button2PreviousState) { // Si cet état a changé
        if (button2currentState == LOW) { // Si cet état est passé de haut à bas (C'est inversé à cause du PullUp)
            Serial.println("Pression bouton n°2"); // C'est que le bouton est pressé, falling edge
        }
        else { // Si cet état est passé de bas à haut
            Serial.println("Dépression bouton n°2"); // C'est que le bouton est relâché, rising edge
            buttonNumber = 2;
        }
        delay(50); // Debounce ?
        button2PreviousState = button2currentState; // Fin de l'action, cet état fait donc désormais parti du passé
    }

    else if (button3currentState != button3PreviousState) { // Si cet état a changé
        if (button3currentState == LOW) { // Si cet état est passé de haut à bas (C'est inversé à cause du PullUp)
            Serial.println("Pression bouton n°3"); // C'est que le bouton est pressé, falling edge
        }
        else { // Si cet état est passé de bas à haut
            Serial.println("Dépression bouton n°3"); // C'est que le bouton est relâché, rising edge
            buttonNumber = 3;
        }
        delay(50); // Debounce ?
        button3PreviousState = button3currentState; // Fin de l'action, cet état fait donc désormais parti du passé
    }

    else { // Revient à dire "Si ni le bouton n°0, ni le n°1, ni le n°2, ni le n°3 n'est pressé c'est que qu'aucun bouton n'est pressé"
        buttonNumber = 4;
    }
    return (buttonNumber); // Lorsque la procédure est executé elle retournera le numéro du bouton pressé contenu dans la variable "buttonNumber"
}

void setup() {
    Serial.begin(9600);
    pinMode(button0, INPUT_PULLUP);
    pinMode(button1, INPUT_PULLUP);
    pinMode(button2, INPUT_PULLUP);
    pinMode(button3, INPUT_PULLUP);
}

void loop() {
    buttonNumber = button(); // Executer la procédure qui va lire l'état des boutons et le retourner
    if (buttonNumber != 4) {
        Serial.print("En gros c'est le: ");
        Serial.println(buttonNumber); // Afficher l'état des boutons
        Serial.println("");
    }
}
```



5.4 Vérifier la séquence de chiffre

Texte

```

13 int button2CurrentStat
14 int button2PreviousStat
15
16 #define button3 6
17 int button3CurrentStat
18 int button3PreviousStat
19
20 int buttonNumber = 0;
21 int SimonSequence[5] =
22
23 int buttonState() {
24 <
```

Moniteur série

```

Depression bouton n1
It's not the right button
Please press a button

Pression bouton n2
Depression bouton n2
It's the right button
Please press a button
```

Utiliser les boutons pour reproduire une séquence de chiffre: [Projet TinkerCad](#)

Une séquence de 5 chiffres est codé en dur, il s'agit de la reproduire en pressant les boutons numérotés de 0 à 3. (Dans un vrai Simon la séquence que le programme doit vérifier fait 31 chiffre)

Ce programme compare la pression de chaque bouton avec l'ordre des chiffres dans la séquence. Si ça correspond, il attend la pression du bouton suivant. Si ça ne correspond pas, il le notifie dans le moniteur série. Une fois la séquence reproduite dans son entièreté, l'utilisateur a « gagné ».

```

#define button0 3
int button0CurrentState = 0;
int button0PreviousState = 1;

#define button1 4
int button1CurrentState = 0;
int button1PreviousState = 1;

#define button2 5
int button2CurrentState = 0;
int button2PreviousState = 1;

#define button3 6
int button3CurrentState = 0;
int button3PreviousState = 1;

int buttonNumber = 0; // Variable contenant le numéro du bouton pressé, 0 signifiant qu'aucun bouton n'est actionné
int SimonSequence[5] = {0, 1, 2, 3, 0}; // Tableau avec 5 cases. La case n°0 contient 0, la case n°1 contient 1 [...] , la case n°4 contient 0
```



```
int buttonState() {
    button0CurrentState = digitalRead(button0); // Récupérer l'état actuel du bouton
    button1CurrentState = digitalRead(button1); // Récupérer l'état actuel du bouton
    button2CurrentState = digitalRead(button2); // Récupérer l'état actuel du bouton
    button3CurrentState = digitalRead(button3); // Récupérer l'état actuel du bouton

    if (button0CurrentState != button0PreviousState) { // Si cet état a changé
        if (button0CurrentState == LOW) { // Si cet état est passé de haut à bas (C'est inversé à
cause du PullUp)
            Serial.println("Pression bouton n0"); // C'est que le bouton est pressé, falling edge
        }
        else { // Si cet état est passé de bas à haut
            Serial.println("Depression bouton n0"); // C'est que le bouton est relaché, rising edge
            buttonNumber = 0;
        }
        delay(50); // Debounce ?
        button0PreviousState = button0CurrentState; // Fin de l'action, cet état fait donc
désormais parti du passé
    }

    else if (button1CurrentState != button1PreviousState) { // Si cet état a changé
        if (button1CurrentState == LOW) { // Si cet état est passé de haut à bas (C'est inversé à
cause du PullUp)
            Serial.println("Pression bouton n1"); // C'est que le bouton est pressé, falling edge
        }
        else { // Si cet état est passé de bas à haut
            Serial.println("Depression bouton n1"); // C'est que le bouton est relaché, rising edge
            buttonNumber = 1;
        }
        delay(50); // Debounce ?
        button1PreviousState = button1CurrentState; // Fin de l'action, cet état fait donc
désormais parti du passé
    }

    else if (button2CurrentState != button2PreviousState) { // Si cet état a changé
        if (button2CurrentState == LOW) { // Si cet état est passé de haut à bas (C'est inversé à
cause du PullUp)
            Serial.println("Pression bouton n2"); // C'est que le bouton est pressé, falling edge
        }
        else { // Si cet état est passé de bas à haut
            Serial.println("Depression bouton n2"); // C'est que le bouton est relaché, rising edge
            buttonNumber = 2;
        }
        delay(50); // Debounce ?
        button2PreviousState = button2CurrentState; // Fin de l'action, cet état fait donc
désormais parti du passé
    }
}
```



```
else if (button3currentState != button3PreviousState) { // Si cet état a changé
    if (button3currentState == LOW) { // Si cet état est passé de haut à bas (C'est inversé à cause du PullUp)
        Serial.println("Pression bouton n3"); // C'est que le bouton est pressé, falling edge
    }
    else { // Si cet état est passé de bas à haut
        Serial.println("Depression bouton n3"); // C'est que le bouton est relaché, rising edge
        buttonNumber = 3;
    }
    delay(50); // Debounce ?
    button3PreviousState = button3currentState; // Fin de l'action, cet état fait donc désormais parti du passé
}

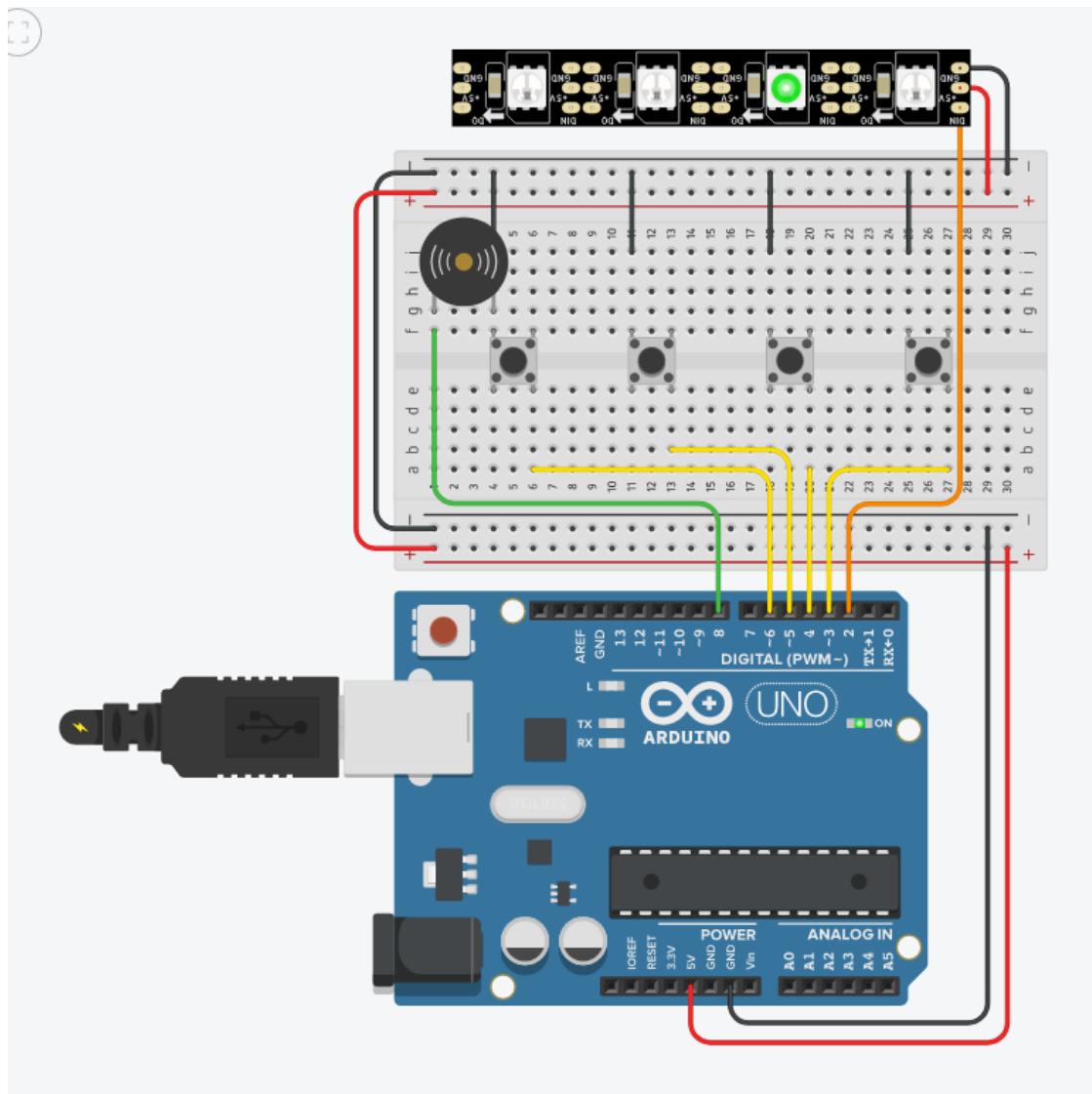
else { // Revient à dire "Si ni le bouton n°0, ni le n°1, ni le n°2, ni le n°3 n'est pressé c'est que qu'aucun bouton n'est pressé"
    buttonNumber = 4;
}
return (buttonNumber); // Lorsque la procédure est executé elle retournera le numéro du bouton pressé contenu dans la variable "buttonNumber"
}

void SimonCheck() {
    Serial.println("Please press a button"); // Demander à l'utilisateur de presser un bouton
    int checkedNumber = 0; // Initialiser une variable permettant de vérifier que tout les chiffres de la séquence ont été pressé
    while (checkedNumber < 5) { // Tant que tout les chiffres de la séquence n'ont pas été pressé
        buttonNumber = buttonState(); // Etre attentif à la pression des boutons
        if (buttonNumber != 4) { // Si un bouton est pressé
            if (buttonNumber == SimonSequence[checkedNumber]) { // Et que c'est le bon au regard de la séquence
                Serial.println("It's the right button"); // Indiquer dans le moniteur série que c'est le bon
                checkedNumber++; // Passer au nombre suivant de la séquence
                delay(100); // Attendre minimum 0.1sec avant la prochaine pression de bouton
            }
            else { // Sinon, c'est que le mauvais bouton est pressé
                Serial.println("It's not the right button"); // Dire à l'utilisateur que ce n'est pas la bonne réponse
                delay(100); // Attendre minimum 0.1sec avant la prochaine pression de bouton
            } // Le laisser réessayer, encore et encore.
            Serial.println("Please press a button"); // Demander à l'utilisateur de pressé un bouton
            Serial.println();
        }
    }
}

void setup() {
    Serial.begin(9600);
    pinMode(button0, INPUT_PULLUP);
    pinMode(button1, INPUT_PULLUP);
    pinMode(button2, INPUT_PULLUP);
    pinMode(button3, INPUT_PULLUP);
}

void loop() {
    SimonCheck(); // Executer la procédure pour vérifier si l'utilisateur a pressé le bon bouton
    Serial.println("You won"); // Indiquer que la vérification est terminé et que l'utilisateur a gagné
    delay(2000); // Lui laisser le temps de se remettre de ses émotions
}
```

5.5 Un Simon basique mais complet



La consécration ! [Projet TinkerCad](#)

Voici donc un Simon tout à fait fonctionnel. Il joue une séquence qui s'allonge et vérifier que la pression des boutons corresponds. Il joue un son si l'on se trompe et recommence la partie de 0. (La séquence est courte pour faciliter les essais lors de la programmation)

Il manque encore quelques fonctionnalités pour coller à l'original :

- Accélération de la cadence d'affichage en fonction de la progression
- Perdre si l'on hésite à presser un bouton pendant plus de 3 secondes
- Créer des niveaux plus facile à gagner, 31 chiffres c'est vraiment long.
- Le fameux mode multijoueur !