

---

# Ressources pour la carte Micro:bit

***Version 1***

**IREM Marseille**

**juin 20, 2018**



---

## Prise en main

---

<b>1</b>	<b>Prise en main de Micro :bit</b>	<b>3</b>
<b>2</b>	<b>Projets à réaliser</b>	<b>35</b>
<b>3</b>	<b>Index et page de recherche</b>	<b>59</b>
<b>Index</b>		<b>61</b>



par le groupe InEFLP de l'IREM de Marseille

### Contenu du site

Vous trouverez dans ce document des ressources permettant de se former à Micro :bit.

### Qui sommes-nous ?

Nous sommes des enseignants de maths/sciences regroupés au sein d'un groupe de recherche de l'IREM de Marseille.

Notre groupe, *Innovation, Expérimentation et Formation en Lycée Professionnel* (InEFLP) a une partie de son travail consacrée l'enseignement de l'algorithme en classes de lycée professionnel. Dans le cadre de cette recherche, nous explorons les objets connectés tels que Arduino ou Microbit.

Site du groupe InEFLP.



Table des matières du document



# CHAPITRE 1

---

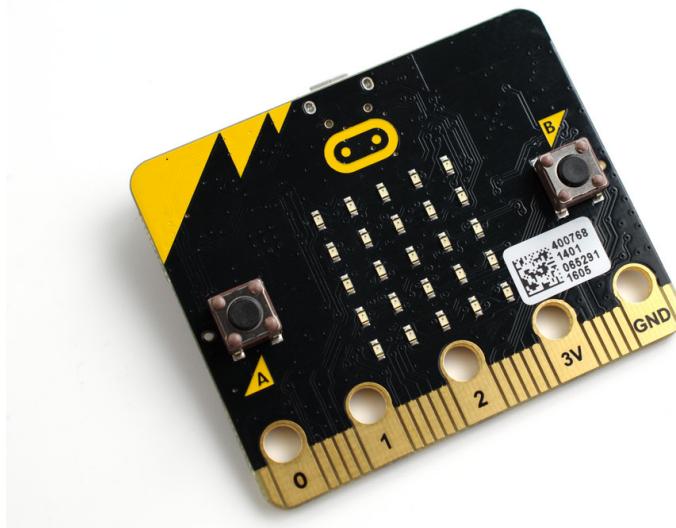
## Prise en main de Micro :bit

---

### 1.1 Micro :bit, c'est quoi ?

Micro :bit est un microcontrôleur développé au Royaume-Unis. Par ses caractéristiques techniques et ses interfaces pédagogiques, cet objet possède un fort potentiel pour l'enseignement de l'algorithme.

Après un bref rappel historique, nous expliquerons plus en détail les caractéristiques propres de cet objet. Nous mettrons ensuite en avant la facilité de mise en œuvre en formation puis nous poursuivrons en donnant un premier aperçu de l'intérêt pédagogique de Micro :bit.



#### 1.1.1 Bref historique

Le développement de Micro :bit s'inscrit dans le cadre d'une politique volontariste de développement de l'apprentissage de la programmation. L'objectif premier visait à équiper tous les élèves de 11/12 ans du Royaume-Unis ainsi que leur

enseignant. Maintenant que c'est chose faite, le reste du monde peut en profiter aussi.

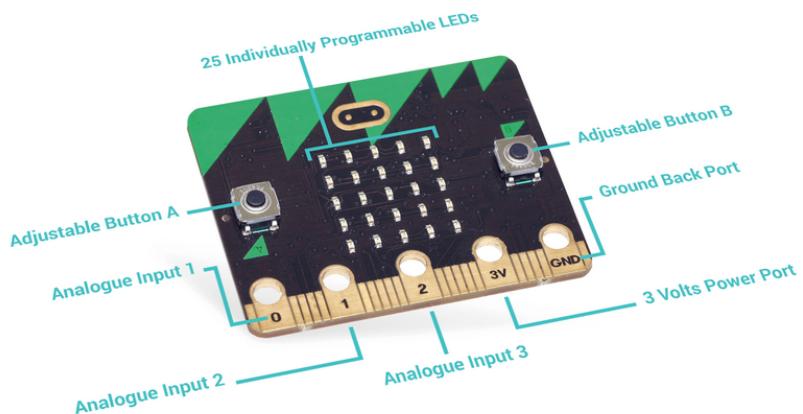


La BBC<sup>1</sup> est le moteur de ce projet. 30 ans après sa première distribution d'ordinateurs aux enfants britanniques<sup>2</sup>, “la Vieille Dame” remet ça aujourd’hui. La BBC utilise ses moyens de diffusions pour promouvoir et accompagner les utilisateurs, notamment en proposant des émissions de TV dédié à cet objet sur un mode ludique et divertissant. Sur les 29<sup>3</sup> partenaires de ce projet, se trouvent entre autres Microsoft<sup>4</sup> pour une partie logiciel et interface de programmation, ARM<sup>5</sup> pour la construction des processeur et la partie matériel, et Samsung<sup>6</sup> pour un support mobile. C'est donc un projet qui mobilise des acteurs majeurs du numériques et de la communication, prévu pour durer.

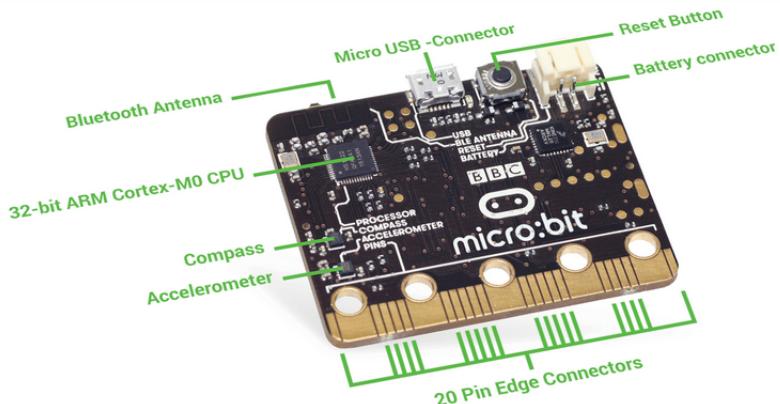
1. Make It Digital - The BBC micro :bit. (s. d.). Consulté 29 mars 2017, à l'adresse <http://www.bbc.co.uk/programmes/articles/4hVG2Br1W1LKcmw8nSm9WnQ/the-bbc-micro-bit>
2. BBC Micro. (2016, septembre 20). In Wikipédia. Consulté à l'adresse [https://fr.wikipedia.org/w/index.php?title=BBC\\_Micro&oldid=129763631](https://fr.wikipedia.org/w/index.php?title=BBC_Micro&oldid=129763631)
3. Partners. (s. d.). Consulté 29 mars 2017, à l'adresse <https://www.microbit.co.uk/partners>
4. The BBC micro :bit and Microsoft - Microsoft Research. (s. d.). Consulté 29 mars 2017, à l'adresse <https://www.microsoft.com/en-us/research/project/the-bbc-microbit-and-microsoft/>
5. Ltd, A. R. M. (s. d.). ARM | Innovation Hub - BBC micro :bit. Consulté 29 mars 2017, à l'adresse <http://www.arm.com/innovation/products/microbit.php>
6. Code on the go with Samsung & micro :bit. (s. d.). Consulté 29 mars 2017, à l'adresse <http://www.samsung.com/uk/citizenship/bbcmicrobit.html>



### 1.1.2 La carte Micro :bit



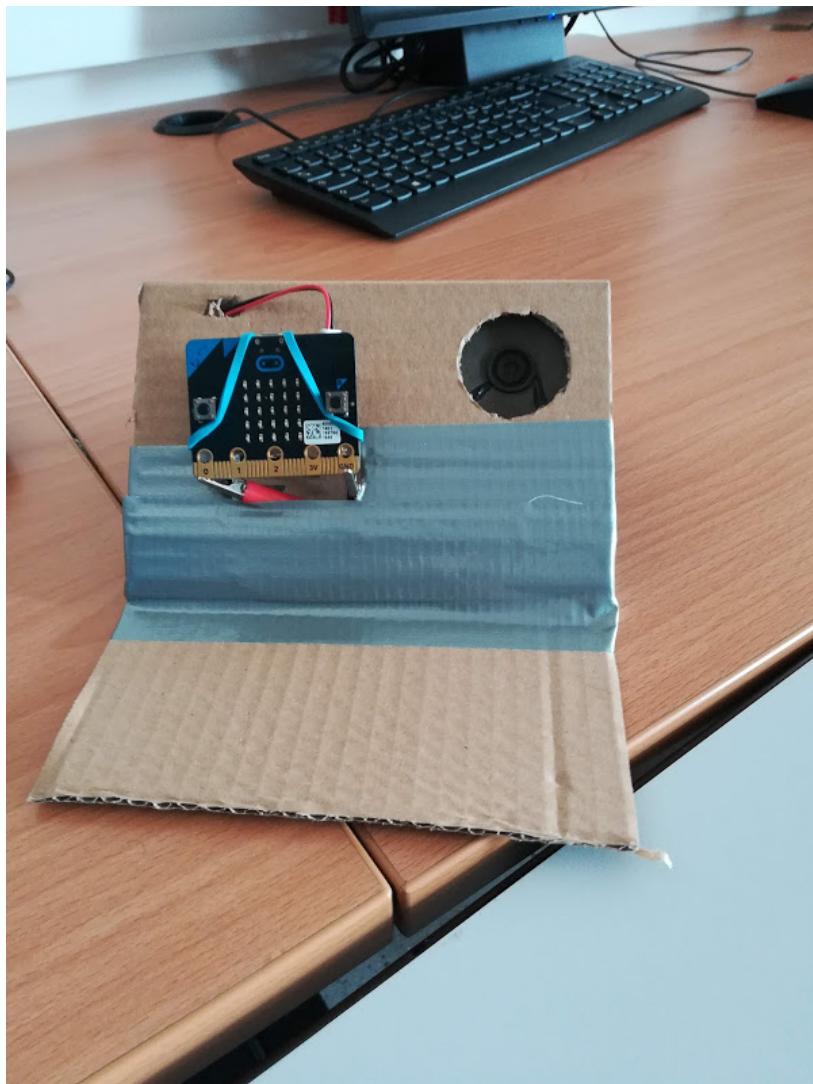
Concrètement de quoi s'agit-il ? On parle ici de microcontrôleur, à savoir une carte électronique programmable pour interagir avec le monde réel. C'est une version accessible de l'électronique que tout un chacun manipule au quotidien sans se poser de question, par exemple les dispositifs de domotique qui permettent de gérer à distance le chauffage, la sécurité, l'arrosage du géranium... Ou bien plus simplement la bouilloire programmable au degré °C près, la guirlande du sapin qui clignote au rythme de "Jingle Bells". Ce microcontrôleur permet d'élaborer par exemple un podomètre, un doudou sensoriel, un sismographe rudimentaire...



L'interface de programmation est conçue pour être utilisable par un enfant d'une dizaine d'année, c'est donc la simplicité qui prime. On dispose en première approche d'une application internet utilisant le principe de la programmation par bloc, à savoir sur le principe des Blockly que l'on retrouve dans Scratch ou StudioCode. En plus d'une programmation accessible, l'interface propose une simulation de la carte. Ceci permet de voir directement les effets du programme dans l'interface. Pour un usage plus avancé il est notamment possible de programmer avec le langage Python<sup>7</sup> ou Javascript.

---

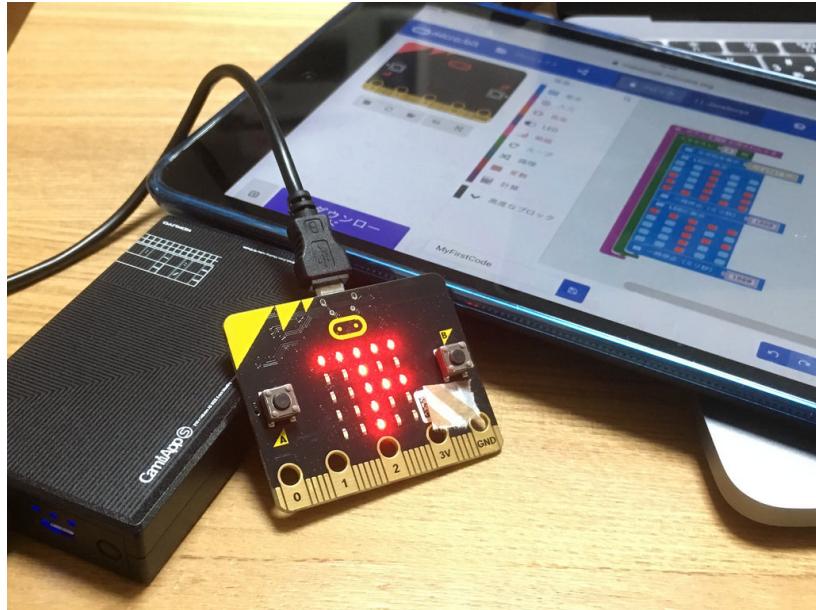
7. Python editor. (s. d.). Consulté 29 mars 2017, à l'adresse <http://python.microbit.org/editor.html>



Bien entendu de nombreux exemples de projets existent, qu'ils soient issus des émissions BBC ou de la communauté éducative. Sur le site officiel on trouve des idées, des tutoriels, des leçons<sup>8</sup> comme par exemple : une alarme de trousse, un compteur de frappe (au baseball) ou encore des leçons sur l'accélération.

---

8. Idées | micro :bit. (s. d.). Consulté 29 mars 2017, à l'adresse <http://microbit.org/fr/ideas/>



### Notes

## 1.2 Programmation par blocs : Makecode

### 1.2.1 Une interface en ligne

L'interface de programmation par blocs a été développée en partenariat avec microsoft, elle se trouve en ligne à cette adresse : <https://makecode.microbit.org/>

Il s'agit donc d'une page internet mais dont le code est mis en cache par le navigateur ce qui signifie qu'elle reste opérationnelle hors ligne.

---

**Note :** A partir de chrome par exemple, il est possible de créer un raccourci sur le bureau.

---

### 1.2.2 Un simulateur !

Le très gros intérêt de cette interface consiste en son simulateur de carte qui permet d'avoir un aperçu du fonctionnement du programme avant même de le télécharger sur la carte.

---

**Note :** Le simulateur peut ne pas fonctionner hors ligne.

---

### 1.2.3 Compilation et enregistrement

Le téléchargement sur la carte se fait très simplement puisqu'elle est reconnue comme une clé USB il suffit donc de cliquer sur « »télécharger»» et de copier le fichier obtenu (.hex) sur la carte.

## 1.2.4 Programmation

Comme toute interface de programmation par blocs, elle est très intuitive à manipuler. Les premiers programmes se font très simplement et les catégories sont classées par couleurs et par technicité.

## 1.2.5 Documentation

Une page de documentation présente les éléments de base pour la programmation par blocs : <https://makecode.microbit.org/blocks>

Une page de références présentent quelques fonctionnalités propre au microbit : <https://makecode.microbit.org/reference>

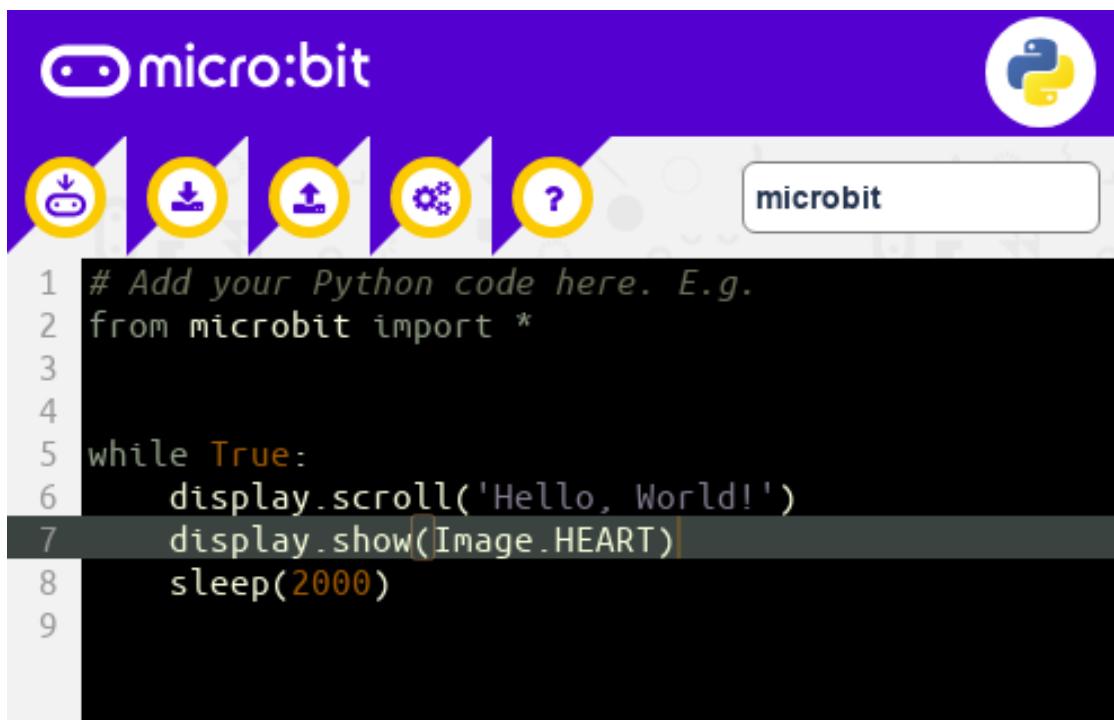
## 1.3 Programmation en Python

Le micro:bit peut exécuter une version allégée de Python qui s'appelle MicroPython. C'est une version spécialement dédiée aux microcontrôleurs.

### 1.3.1 Une interface en ligne

Il est possible de programmer en python à partir d'un éditeur en ligne <http://python.microbit.org>

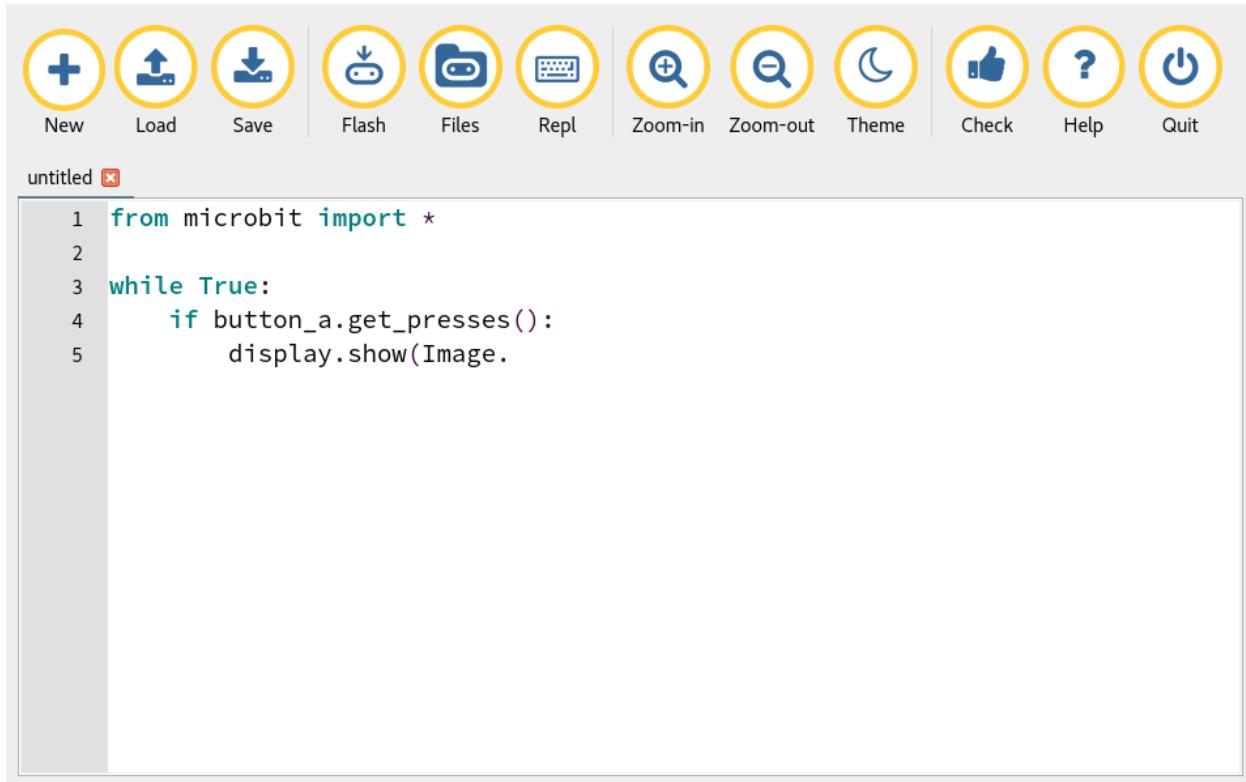
L'interface est cependant assez pauvre en fonctionnalité et ne dispose pas de l'autocomplétion.



```
1 # Add your Python code here. E.g.
2 from microbit import *
3
4
5 while True:
6     display.scroll('Hello, World!')
7     display.show(Image.HEART)
8     sleep(2000)
9
```

### 1.3.2 Mu : une interface complète

Comme le dit (en anglais) la page d'accueil de Mu : *Mu est un éditeur de code simple pour les programmeurs débutants. Il est développé en Python et fonctionne sur Windows, OSX, Linux et Raspberry Pi.*



### 1.3.3 Programmation

L'autocomplétion et l'autoindentation est très efficace. L'interface est rapidement utilisable par un débutant en programmation.

### 1.3.4 Compilation et enregistrement

Le téléchargement sur la carte se fait très simplement puisqu'il suffit de cliquer sur le bouton . Il est tout de même préférable d'avoir au préalable le réflexe de vérifier le code avec .

### 1.3.5 Communication série

La fonction REPL de Mu permet d'ouvrir une communication via un port série avec le micro :bit. Il est ainsi possible d'envoyer et de recevoir des données. Sur les versions bêta il y a même un plotteur qui permet de visualiser graphiquement les données reçues.

### 1.3.6 Documentation

Il existe une documentation sur microbit et micropython, qui bien qu'en anglais reste très accessible.

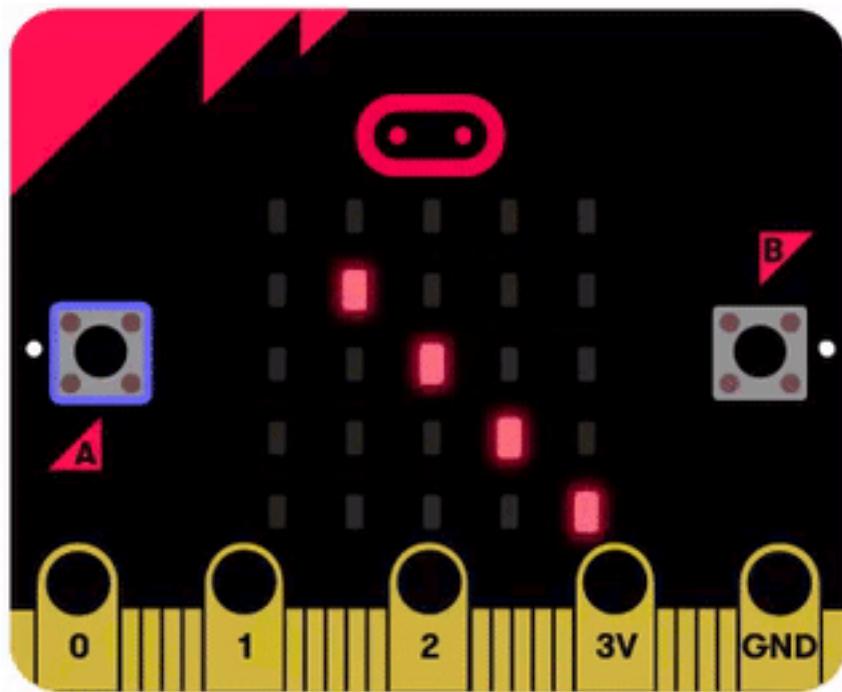
<https://microbit-micropython.readthedocs.io/en/latest/index.html>

## 1.4 Pile ou face (blocs)

### 1.4.1 Description

Le but de ce projet de simuler une expérience aléatoire de lancer de pièce. A partir d'une situation simple idéale pour une prise en main de l'interface de programmation, il s'agit par la suite d'améliorer le programme pas à pas. L'objectif est d'obtenir un programme utilisable dans le cadre de d'un cours sur les statistiques et les probabilités.

On utilise l'interface de programmation par bloc : <https://makecode.microbit.org/>



### Exemple(s) d'utilisation

- Algorithmique et programmation (thème E) du programme de cycle 4
- Domaine statistiques et probabilités du programme de mathématiques de Bac Pro
- Accompagnement personnalisé pour des élèves de seconde Bac Pro

### 1.4.2 Progression

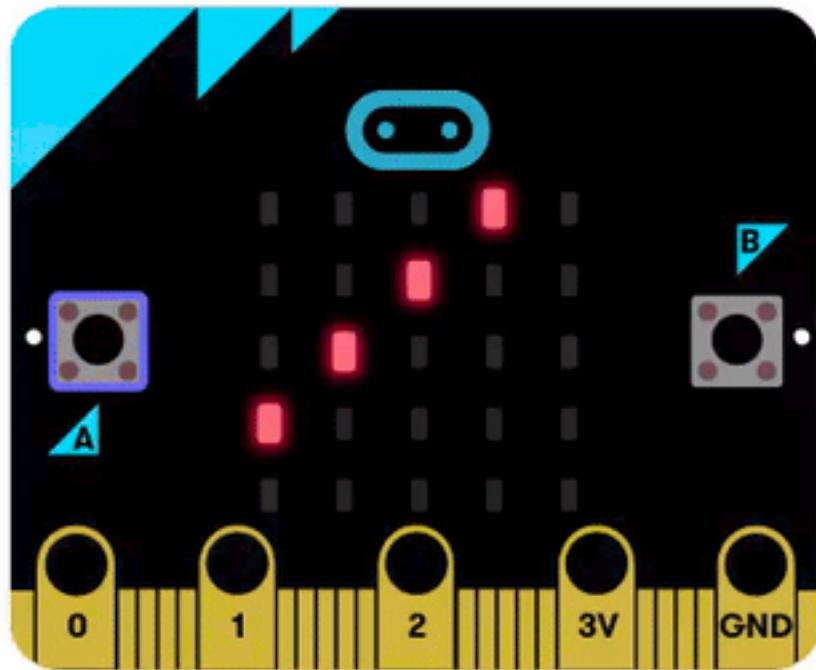
#### Niveau 1

Ce que l'on veut obtenir : afficher 0 ou 1 de façon aléatoire à l'issue d'une courte animation. Ce premier niveau permet de se familiariser avec l'interface tout en produisant un premier programme fonctionnel et utile.

### Les notions abordées

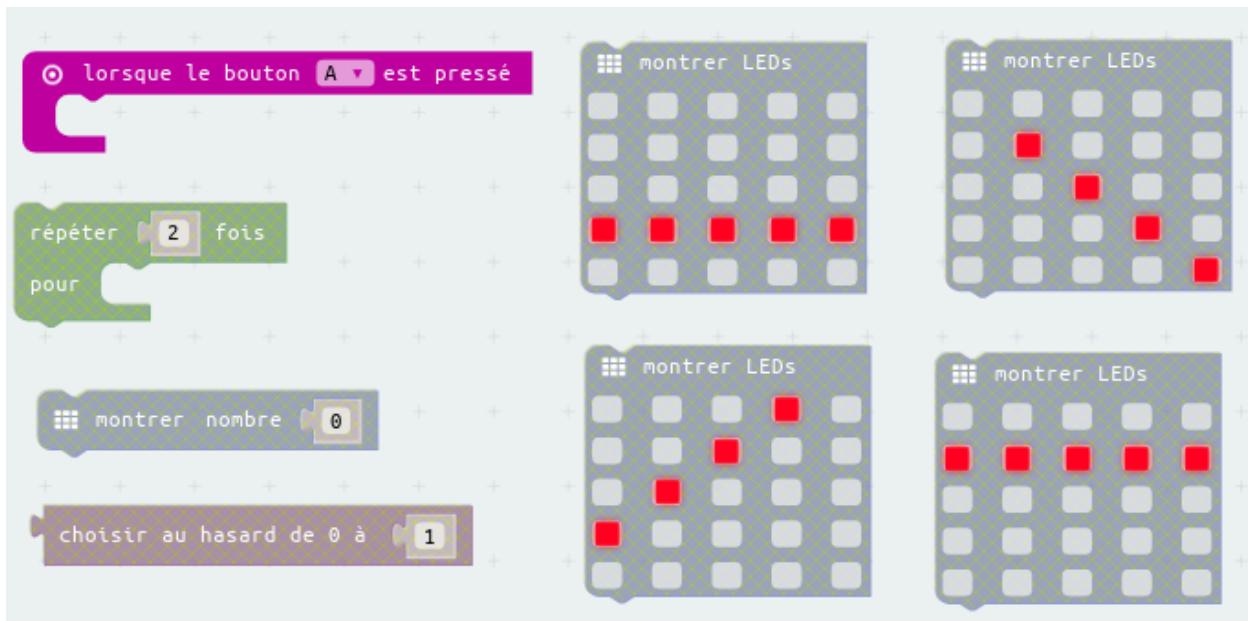
Dans ce niveau nous trouvons les notions suivantes :

- interactions avec l'utilisateur (bouton, affichage)
- boucle
- aléa



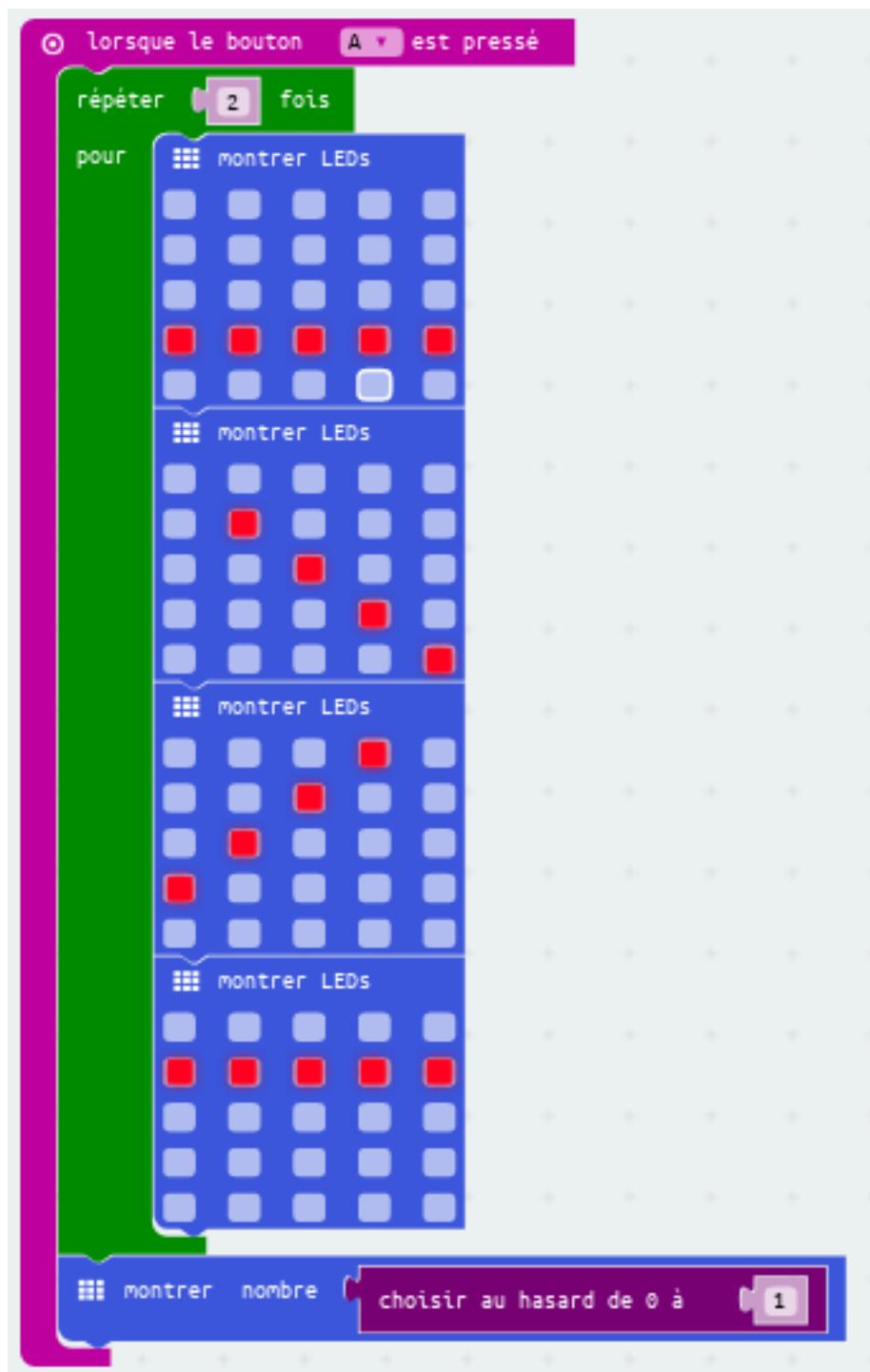
### Les blocs préconisés

On propose aux élèves d'utiliser les blocs suivant :



### Une solution possible

Le résultat escompté est le suivant :



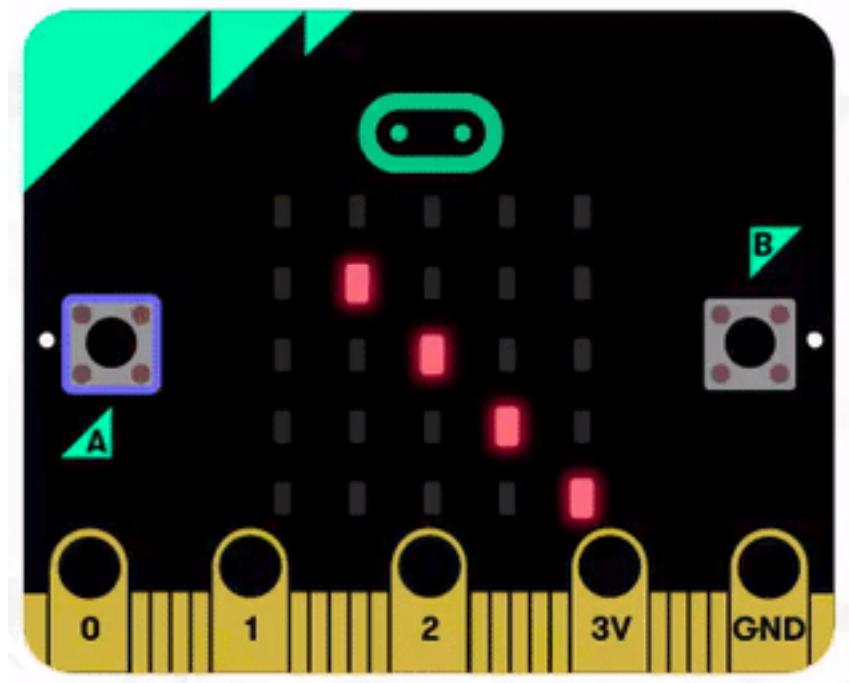
## Niveau 2

Ce que l'on veut obtenir : afficher P ou F de façon aléatoire à l'issue d'une courte animation. Pour ce deuxième niveau, on va utiliser deux nouveaux blocs seulement.

### Les notions abordées

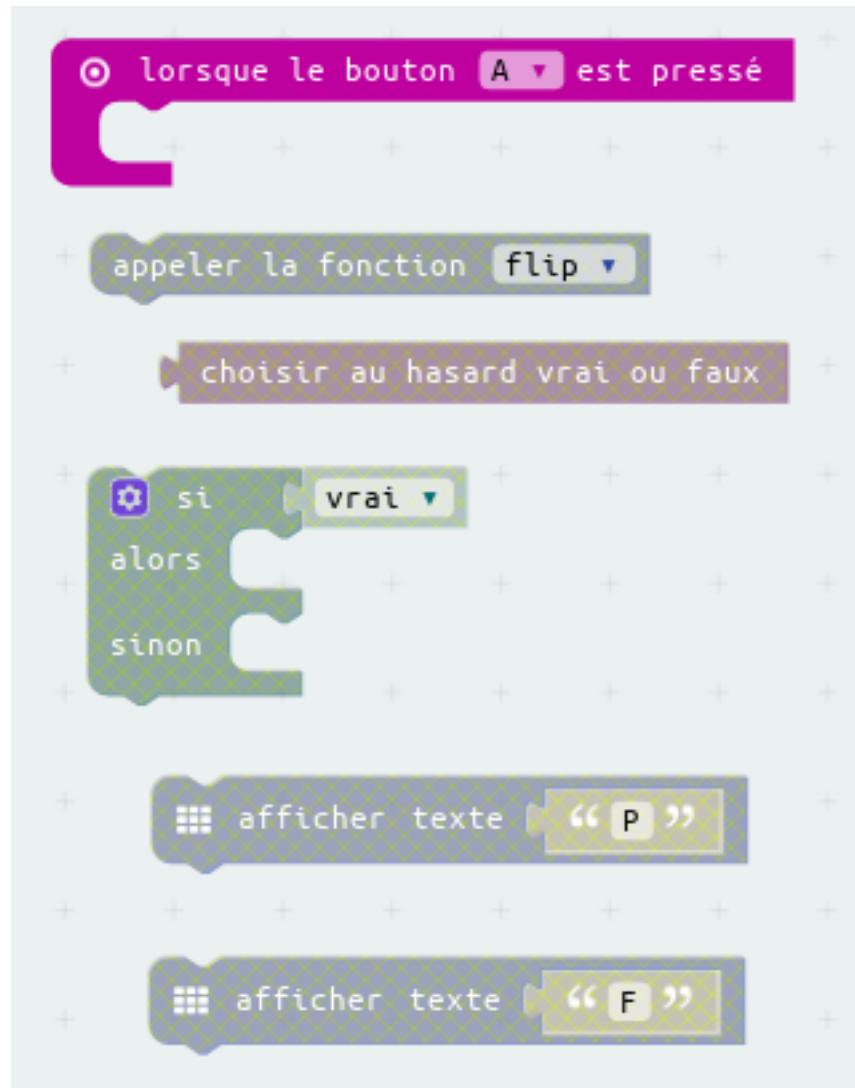
Dans ce niveau nous trouvons les notions suivantes :

- fonction (création et appel)
- instruction conditionnelle



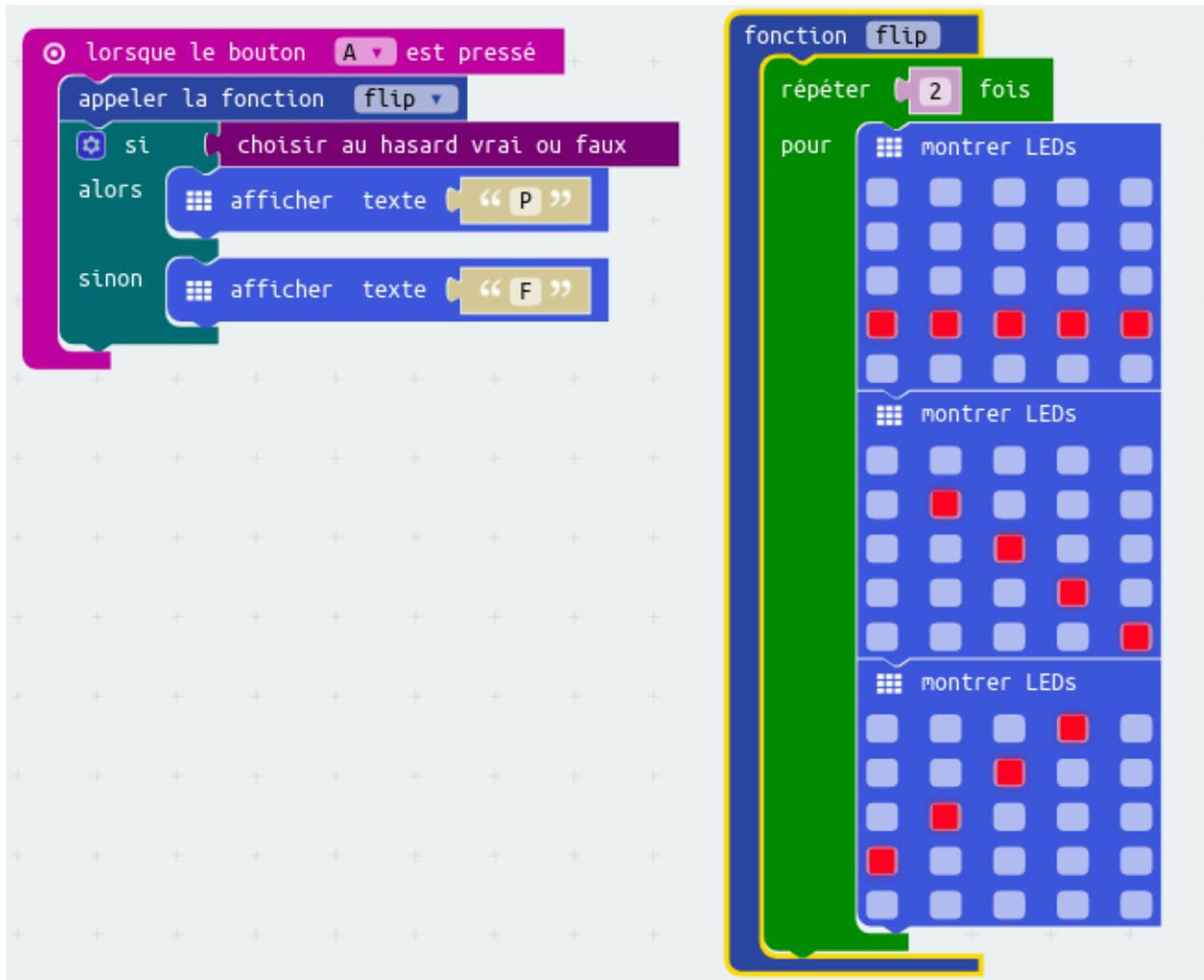
### Les blocs préconisés

On propose aux élèves d'utiliser les blocs suivant :



#### Une solution possible

Le résultat escompté est le suivant :



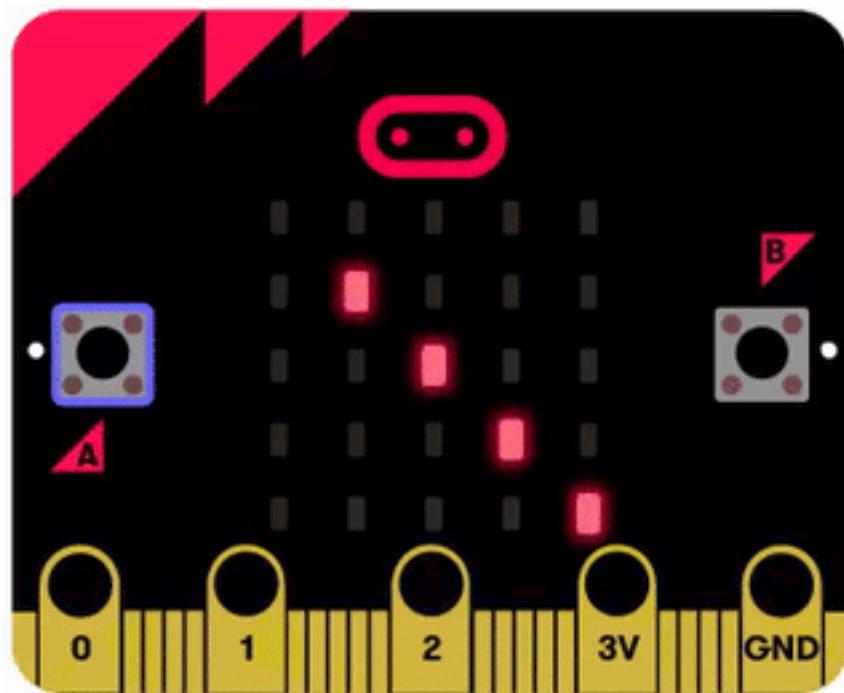
### Niveau 3

Ce que l'on veut obtenir : compter les issues obtenues et afficher le résultat. Pour parvenir à cela il va falloir utiliser une variable.

#### Les notions abordées

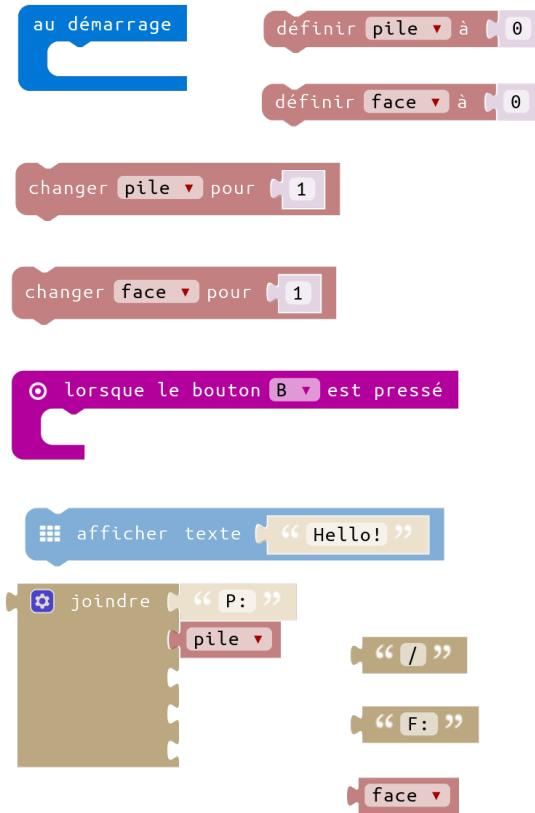
Dans ce niveau nous trouvons les notions suivantes :

- définition d'une variable
- incrémentation d'une variable
- concaténation de texte et de valeur



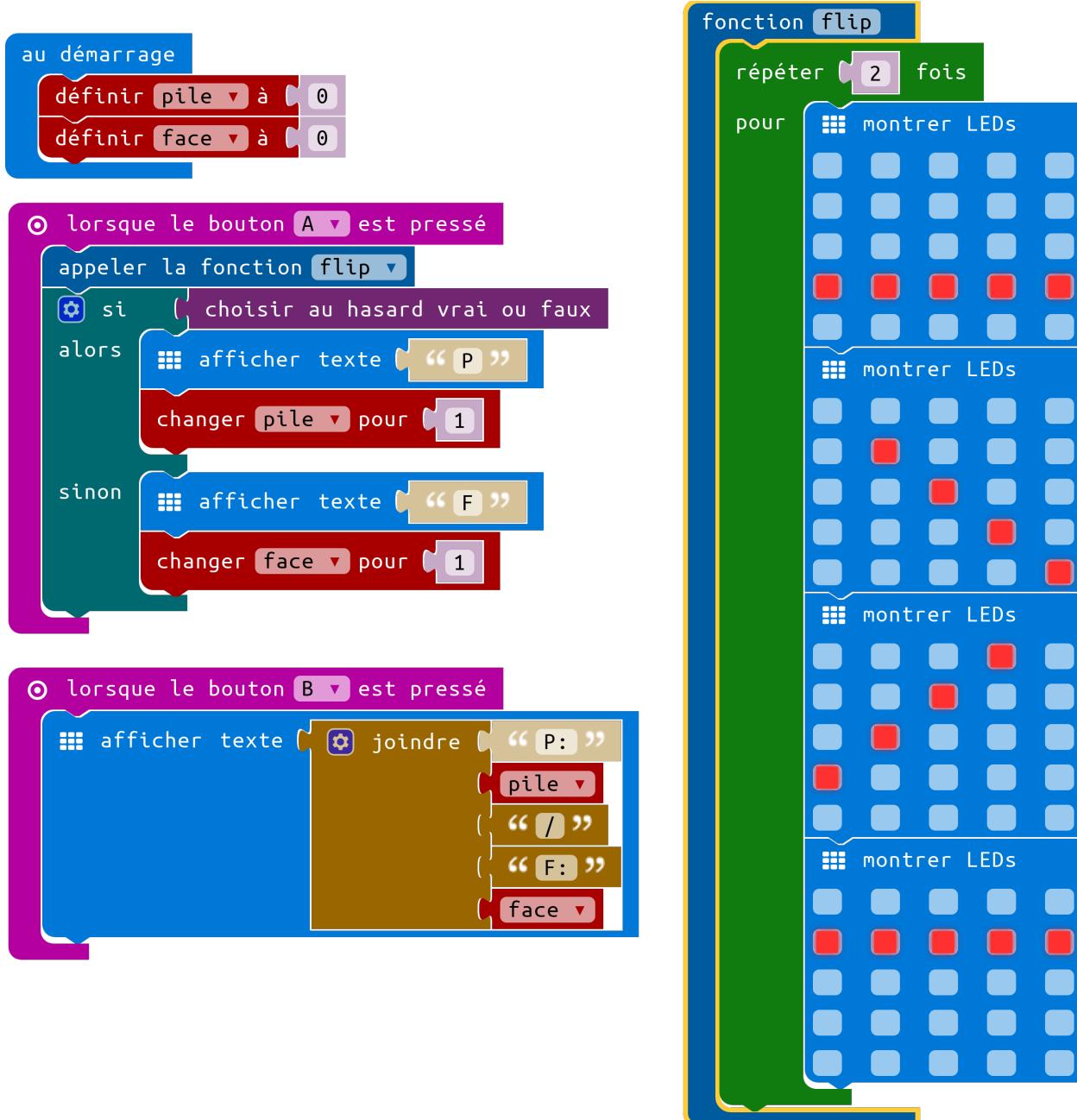
#### Les blocs préconisés

On propose aux élèves d'utiliser les blocs suivant :



### Une solution possible

Le résultat escompté est le suivant :

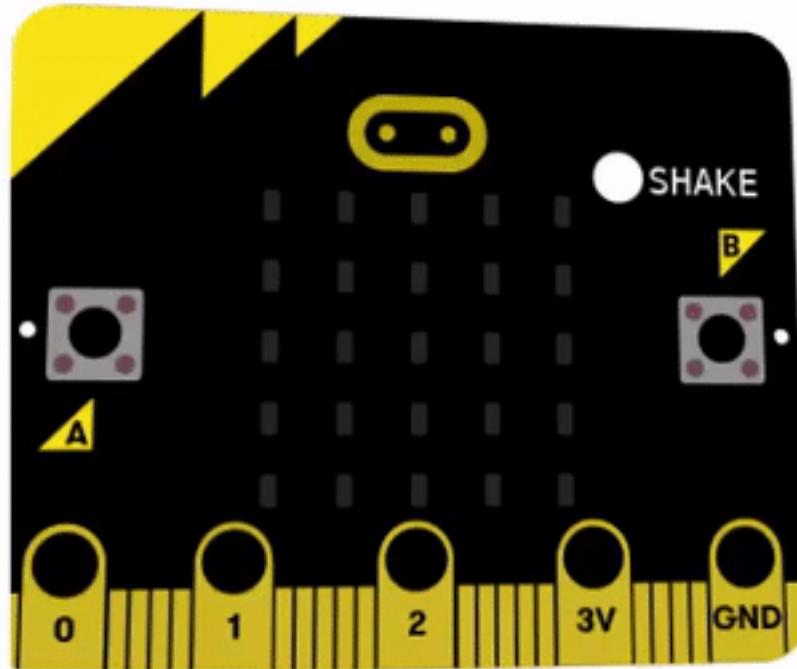


## 1.5 Dé 6 faces (blocs)

### 1.5.1 Description

Le but de ce projet de simuler une expérience aléatoire de lancer d'un dé à 6 faces. Toujours à partir d'une situation simple idéale , le programme peut être étayé au gré des besoins.

On utilise l'interface de programmation par bloc : <https://makecode.microbit.org/>



#### Exemple(s) d'utilisation

- Algorithmique et programmation (thème E) du programme de cycle 4
- Domaine statistique et probabilités du programme de mathématiques en seconde et première Bac Pro.
- Accompagnement personnalisé pour des élèves de seconde Bac Pro

#### 1.5.2 Progression

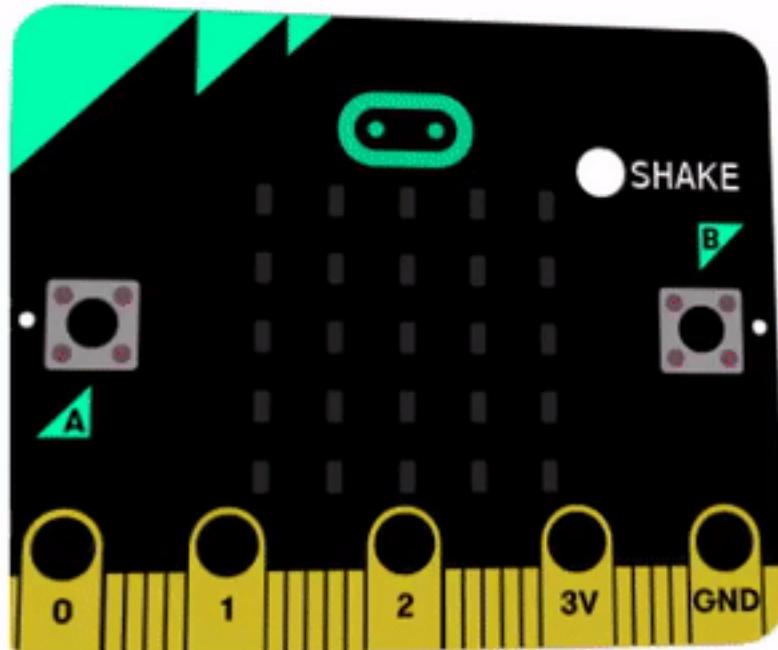
##### Niveau 1

Ce que l'on veut obtenir : afficher un nombre entier entre 1 et 6 de façon aléatoire en secouant l'appareil. Ce premier niveau permet d'introduire la problématique.

##### Les notions abordées

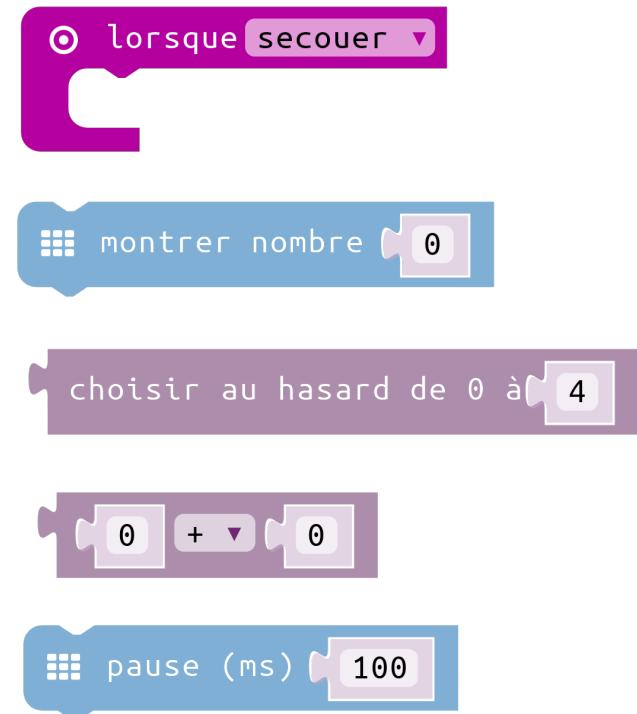
###### Dans ce niveau nous trouvons les notions suivantes :

- interactions avec l'utilisateur (bouton, affichage)
- boucle
- aléa



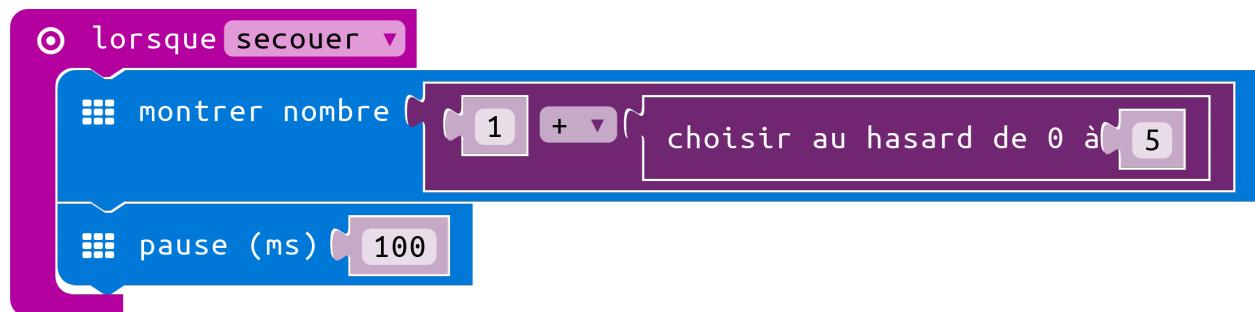
### Les blocs préconisés

On propose aux élèves d'utiliser les blocs suivant :



### Une solution possible

Le résultat escompté est le suivant :



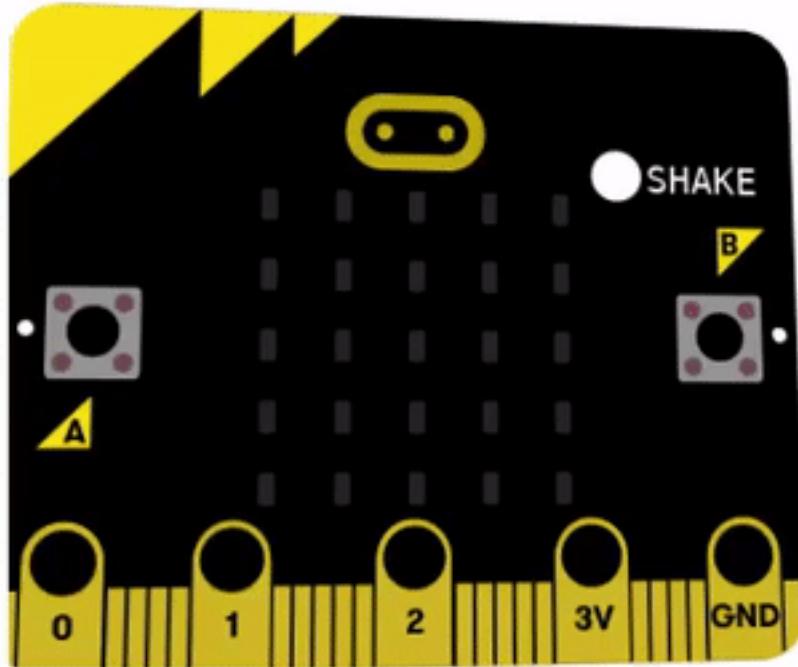
### Niveau 2

Ce que l'on veut obtenir : simuler l'affichage d'un dé à l'issue d'un tirage aléatoire. Ce deuxième niveau, va permettre d'introduire de nouvelle notions.

### Les notions abordées

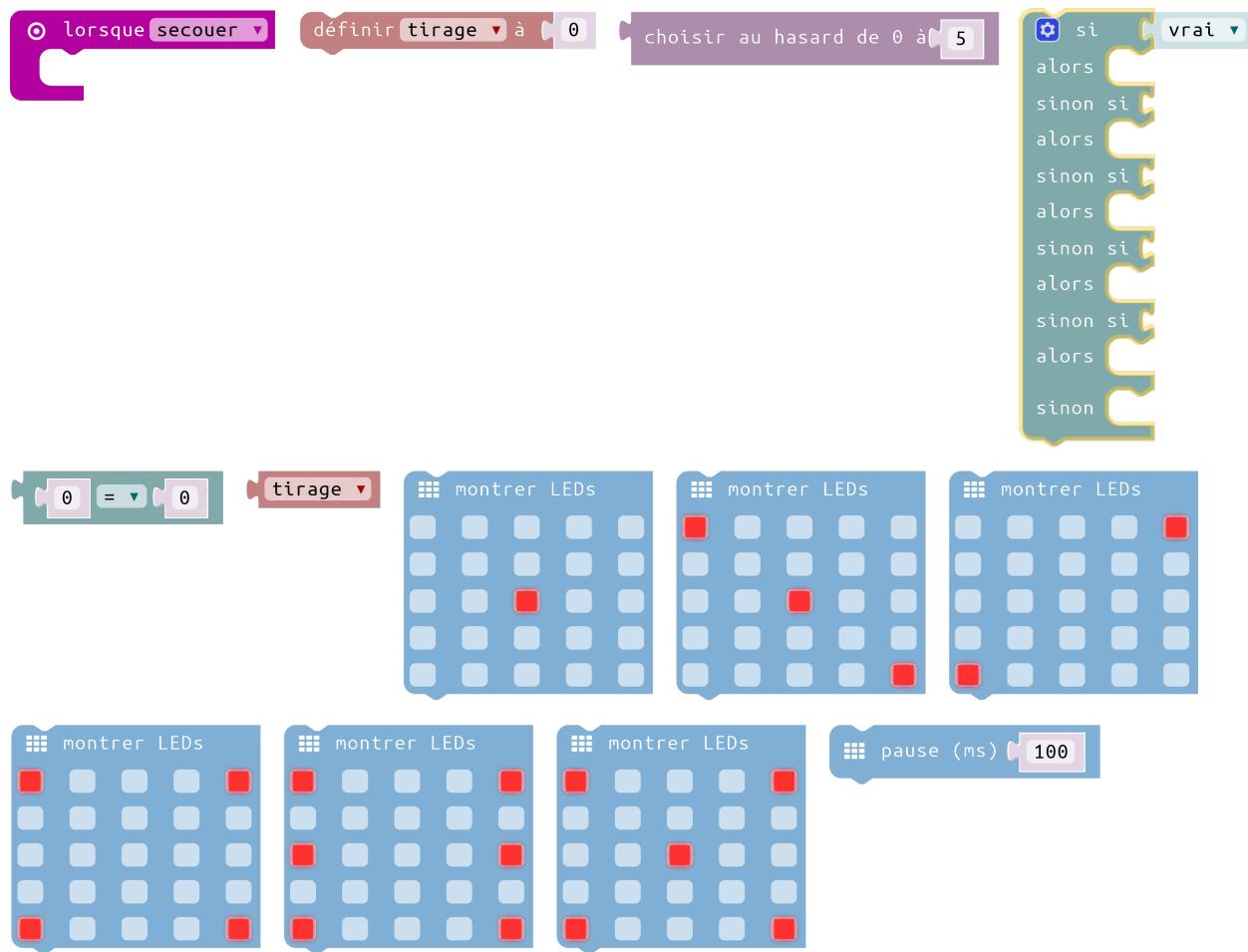
Dans ce niveau nous trouvons les notions suivantes :

- variable
- instruction conditionnelle si/sinon



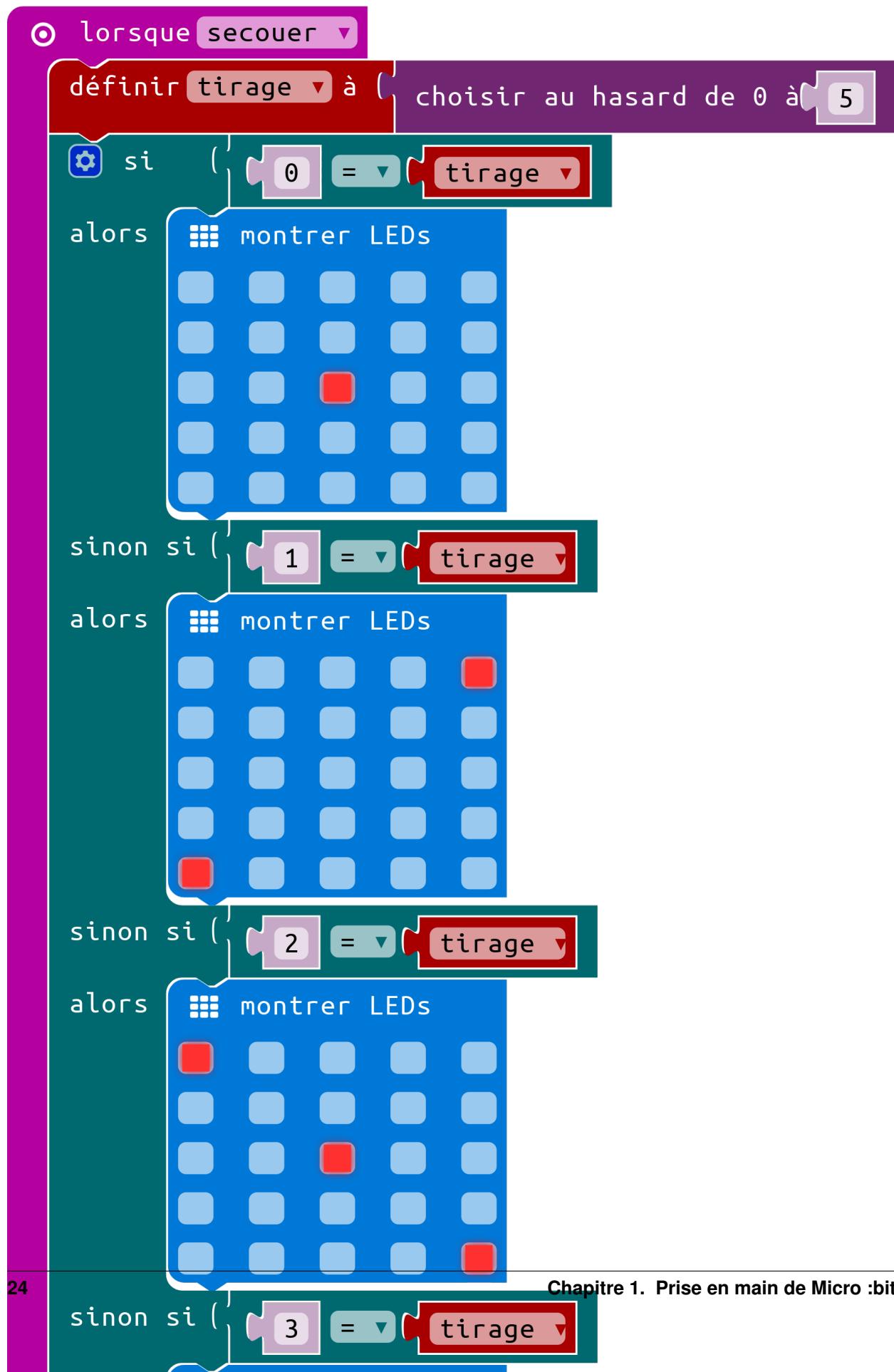
### Les blocs préconisés

On propose aux élèves d'utiliser les blocs suivant :



### Une solution possible

Le résultat escompté est le suivant :



## 1.6 Pile ou face (Python)

### 1.6.1 Description

Le but de ce projet de simuler une expérience aléatoire de lancer de pièce. A partir d'une situation simple idéale pour une prise en main avec Python il s'agit par la suite d'améliorer le programme pas à pas.

*Pile ou face (blocs)*

On verra qu'il ne s'agit pas forcément de la transposition en Python des activités proposées en bloc.

Il est intéressant de relever pour chaque étape l'apport que représente la programmation avec ce langage.

### Exemple(s) d'utilisation

- Domaine statistiques et probabilités du programme de mathématiques de Bac Pro
- Accompagnement personnalisé pour des élèves de seconde Bac Pro
- Algorithmique et programmation au lycée général et technologique

### 1.6.2 Progression

#### Niveau 1

Ce que l'on veut obtenir : afficher « P » ou « F » de façon aléatoire à l'issue d'une courte animation. Ce premier niveau permet de se familiariser avec les fonctions utilisées pour interagir avec le microbit. Contrairement à la programmation par bloc, il est plus efficace ici de choisir « P » ou « F » aléatoirement dans la liste composée de ces 2 singulons. De plus cela permettra facilement de truquer l'expérience aléatoire.

#### Les notions abordées

**Dans ce niveau nous trouvons les notions suivantes :**

- interactions avec le microbit (bouton, affichage)
- aléa (random)
- notion de liste

#### Les éléments utiles

On propose aux élèves d'appeler les éléments suivants

```
import random # bibliothèque pour générer de l'aléa
Image("xxxxx:xxxxx:xxxxx:xxxxx:xxxxx") # où x représente l'intensité d'une diode
→ comprise entre 0 et 9
random.choice(liste) # pour choisir un élément au hasard dans une liste
["P", "F"] # liste des issues (texte) que l'on veut afficher
```

#### Une solution possible

Le résultat escompté est le suivant :

```
from microbit import *

import random

piece1 = Image(          # définition de l'image "piece1"
    "00000:"           # aucune diode n'est éclairée
    "00000:"
    "00000:"
    "99999:"          # toutes les diodes de la 4ème ligne sont éclairées
    ↪ au maximum
    "00000:")

piece2 = Image(
    "00000:"
    "90000:"
    "09000:"
    "00900:"
    "00090:")

piece3 = Image(
    "00000:"
    "00900:"
    "00900:"
    "00900:"
    "00900:")

piece4 = Image(
    "00000:"
    "00009:"
    "00090:"
    "00900:"
    "09000:")

piece5 = Image(
    "00000:"
    "00000:"
    "99999:"
    "00000:"
    "00000:")

while True:
    if button_a.get_presses():
        display.show(piece1)      # la matrice de LED montre l'image "piece1"
        sleep(200)
        display.show(piece2)
        sleep(200)
        display.show(piece3)
        sleep(200)
        display.show(piece4)
        sleep(200)
        display.show(piece5)
        sleep(200)
        display.show(piece1)
        sleep(200)
        display.show(random.choice(["P", "F"])) # affichage au hasard de P ou F
```

## Niveau 2

Ce que l'on veut obtenir : afficher « P » ou « F » de façon aléatoire à l'issue d'une courte animation. L'intérêt ici est de comprendre l'appel à une liste pour l'animation et ainsi de gagner en efficacité et en lisibilité.

### Les notions abordées

Ce niveau permet d'appréhender une utilité supplémentaire du type d'objet « liste ».

### Les éléments utiles

On propose aux élèves d'appeler les éléments suivants

```
[a, b ,c ...] # une liste ou a,b,c ... sont le nom d'images déclarées précédemment
```

### Une solution possible

Le résultat escompté est le suivant :

```
from microbit import *
import random

piece1 = Image(
    "00000:"
    "00000:"
    "00000:"
    "99999:"
    "00000:")

piece2 = Image(
    "00000:"
    "90000:"
    "09000:"
    "00900:"
    "00090:")

piece3 = Image(
    "00000:"
    "00900:"
    "00900:"
    "00900:"
    "00900:")

piece4 = Image(
    "00000:"
    "00009:"
    "00090:"
    "00900:"
    "09000:")

piece5 = Image(
    "00000:"
```

(suite sur la page suivante)

(suite de la page précédente)

```
"00000:"  
"99999:"  
"00000:"  
"00000:")  
  
pieces = [piece1, piece2, piece3, piece4, piece5, piece1]    # la séquence d'images  
  
while True:  
    if button_a.get_presses():  
        display.show(pieces, delay=200)                # la matrice affiche chacune des images de la liste "pieces" avec une pause de 200ms entre chaque image  
        display.show(random.choice(["P", "F"]))
```

### Niveau 3

Ce que l'on veut obtenir : afficher « P » ou « F » de façon aléatoire à l'issue d'une courte animation et compter le nombre d'issues obtenues.

#### Les notions abordées

Pour ce niveau, on va avoir besoin :

- d'une variable pour stocker le résultat du tirage
- de variables pour dénombrer les issues « P » et les issues « F »
- d'une instruction conditionnelle pour tester et agir selon le résultat du tirage

#### Les éléments utiles

On propose aux élèves d'appeler les éléments suivants

```
if: et else: # instructions conditionnelle  
==           # qui permet de vérifier l'égalité entre deux objets  
+= 1         # qui permet d'incrémenter une variable de 1
```

#### Une solution possible

Le résultat escompté est le suivant :

```
from microbit import *  
  
import random  
  
p = 0          # variable stockant le nombre d'issues pile  
f = 0          # variable stockant le nombre d'issues face  
  
piece1 = Image("00000:"  
               "00000:"  
               "00000:"  
               "99999:"  
               "00000:")
```

(suite sur la page suivante)

(suite de la page précédente)

```

piece2 = Image("00000:"
               "90000:"
               "09000:"
               "00900:"
               "00090:")

piece3 = Image("00000:"
               "00900:"
               "00900:"
               "00900:"
               "00900:")

piece4 = Image("00000:"
               "00009:"
               "00090:"
               "00900:"
               "09000:")

piece5 = Image("00000:"
               "00000:"
               "99999:"
               "00000:"
               "00000:")

pieces = [piece1, piece2, piece3, piece4, piece5, piece1]

while True:
    if button_a.get_presses():
        display.show(pieces, delay=200)
        issue = random.choice(["P", "F"])
        if issue == "P":
            display.show("P")
            p += 1
            # incrémentation de la variable p (pile)
        else:
            display.show("F")
            f += 1
            # incrémentation de la variable f (face)

    if button_b.get_presses():
        display.scroll("P:"+str(p))      # affichage du nombre d'issues associées à P
        delay = 200
        display.scroll("F:"+str(f))      # affichage du nombre d'issues associées à F

```

## 1.7 Dé 6 faces (Python)

### 1.7.1 Description

Le but de ce projet de simuler une expérience aléatoire de lancer d'un dé à 6 faces. Tout comme le projet simulant un pile ou face, cette situation permet de comprendre l'intérêt des listes et va un peu plus loin dans leur utilisation.

#### Exemple(s) d'utilisation

- Algorithmique et programmation au lycée général et technologique
- Domaine statistique et probabilités du programme de mathématiques en seconde et première Bac Pro.

- Accompagnement personnalisé pour des élèves de seconde Bac Pro

### 1.7.2 Progression

Pour afficher aléatoirement un nombre entier entre 1 et 6, on peut se contenter des 3 lignes ci-dessous.

```
while True:  
    if button_a.get_presses():  
        # "str" car "display.show" n'affiche que du texte  
        display.show(str(random.randint(1, 6)))
```

Ce qui n'a pas d'autre intérêt que d'introduire la fonction `randint`. Ici, contrairement à la progression utilisée lors du projet en blocs, nous suggérons de commencer par simuler l'affichage tel qu'il apparaît sur un vrai dé. Cela permet de réexploiter les listes d'images introduites avec le projet pile ou face.

### 1.7.3 Progression

#### Niveau 1

Ce que l'on veut obtenir : afficher un nombre entier entre 1 et 6 de façon aléatoire mais à la façon d'un vrai dé, c-a-d avec des points. Ce premier niveau permet d'introduire la problématique et de réinvestir les notions utilisées lors du projet pile ou face.

#### Les notions abordées

**Dans ce niveau nous trouvons les notions suivantes :**

- constitution d'une liste
- tirage d'un élément au hasard dans une liste

#### Les éléments utiles

On propose aux élèves d'appeler les éléments suivants

```
.. code-block:: python  
  
un = Image("00000:00000:00900:00000:00000")    # création d'une image par face  
issues = [un, ...]      # création d'une liste d'images  
random.choice(issues)    # tirage d'un élément au hasard dans une liste
```

#### Une solution possible

Le résultat escompté est le suivant :

```
from microbit import *  
  
import random  
  
un = Image("00000:00000:00900:00000:00000")  
deux = Image("00009:00000:00000:00000:90000")  
trois = Image("90000:00000:00900:00000:00009")
```

(suite sur la page suivante)

(suite de la page précédente)

```
quatre = Image("90009:00000:00000:00000:90009")
cinq = Image("90009:00000:00900:00000:90009")
six = Image("90009:00000:90009:00000:90009")

issues = [un, deux, trois, quatre, cinq, six]

while True:
    if button_a.get_presses():
        display.show(random.choice(issues))
        sleep(800)
        display.clear()
```

## Niveau 2

Ce que l'on veut obtenir :

### Les notions abordées

Dans ce niveau nous trouvons les notions suivantes :

—

### Les éléments utiles

On propose aux élèves d'appeler les éléments suivants

### Une solution possible

Le résultat escompté est le suivant :

```
from microbit import *

import random

un = Image("00000:00000:00900:00000:00000")
deux = Image("00009:00000:00000:00000:90000")
trois = Image("90000:00000:00900:00000:00009")
quatre = Image("90009:00000:00000:00000:90009")
cinq = Image("90009:00000:00900:00000:90009")
six = Image("90009:00000:90009:00000:90009")

n1 = 0
n2 = 0
n3 = 0
n4 = 0
n5 = 0
n6 = 0

issues = [un, deux, trois, quatre, cinq, six]

while True:
    if button_a.get_presses():
```

(suite sur la page suivante)

(suite de la page précédente)

```
issue = random.choice(issues)
display.show(issue)
if issue == un:
    n1 += 1
elif issue == deux:
    n2 += 2
elif issue == trois:
    n3 += 1
elif issue == quatre:
    n4 += 1
elif issue == cinq:
    n5 += 1
elif issue == six:
    n6 += 1
sleep(1000)
display.clear()

if button_b.get_presses():
    display.scroll(
        str(n1)+" /"+str(n2)+" /"
        + str(n3)+" /"+str(n4)+" /"
        + str(n5)+" /"+str(n6)
    )
```

### Niveau 3

Ce que l'on veut obtenir : afficher un nombre entier entre 1 et 6 de façon aléatoire en secouant l'appareil.

#### Les notions abordées

Dans ce niveau nous trouvons les notions suivantes :

#### Les éléments utiles

On propose aux élèves d'appeler les éléments suivants

#### Une solution possible

Le résultat escompté est le suivant :

```
from microbit import *
import random

un = Image("00000:00000:00900:00000:00000")
deux = Image("00009:00000:00000:00000:90000")
trois = Image("90000:00000:00900:00000:00009")
quatre = Image("90009:00000:00000:00000:90009")
cinq = Image("90009:00000:00900:00000:90009")
six = Image("90009:00000:90009:00000:90009")
```

(suite sur la page suivante)

(suite de la page précédente)

```
n = [0, 0, 0, 0, 0, 0]

issues = [un, deux, trois, quatre, cinq, six]

while True:
    if button_a.get_presses():
        i = random.randint(0, 5)
        display.show(issues[i])
        n[i] += 1
        sleep(1000)
        display.clear()

    if button_b.get_presses():
        for k in range(5):
            display.scroll(str(n[k]) + " / ")
```



# CHAPITRE 2

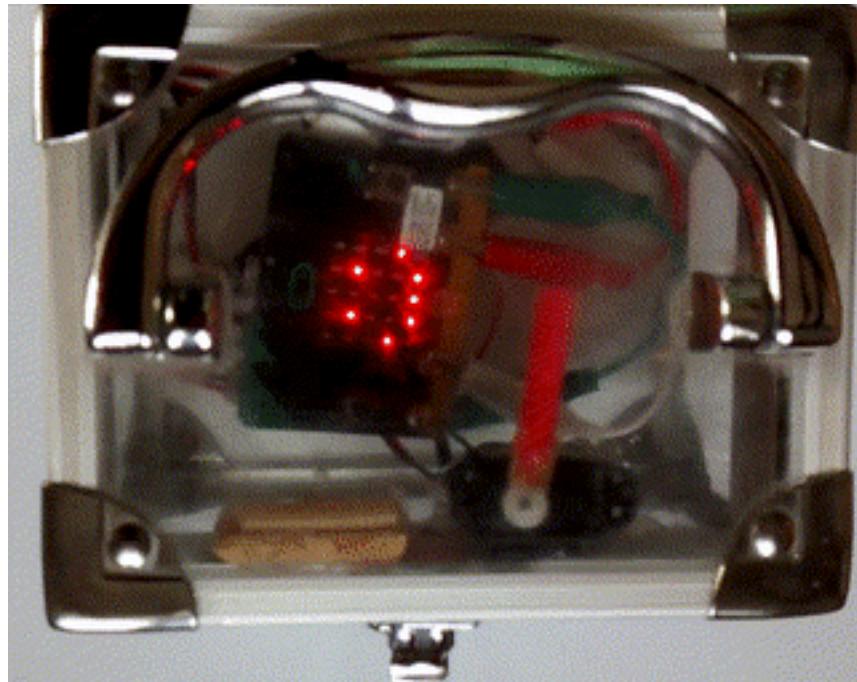
---

Projets à réaliser

---

## 2.1 Boîte fermée

### 2.1.1 Description



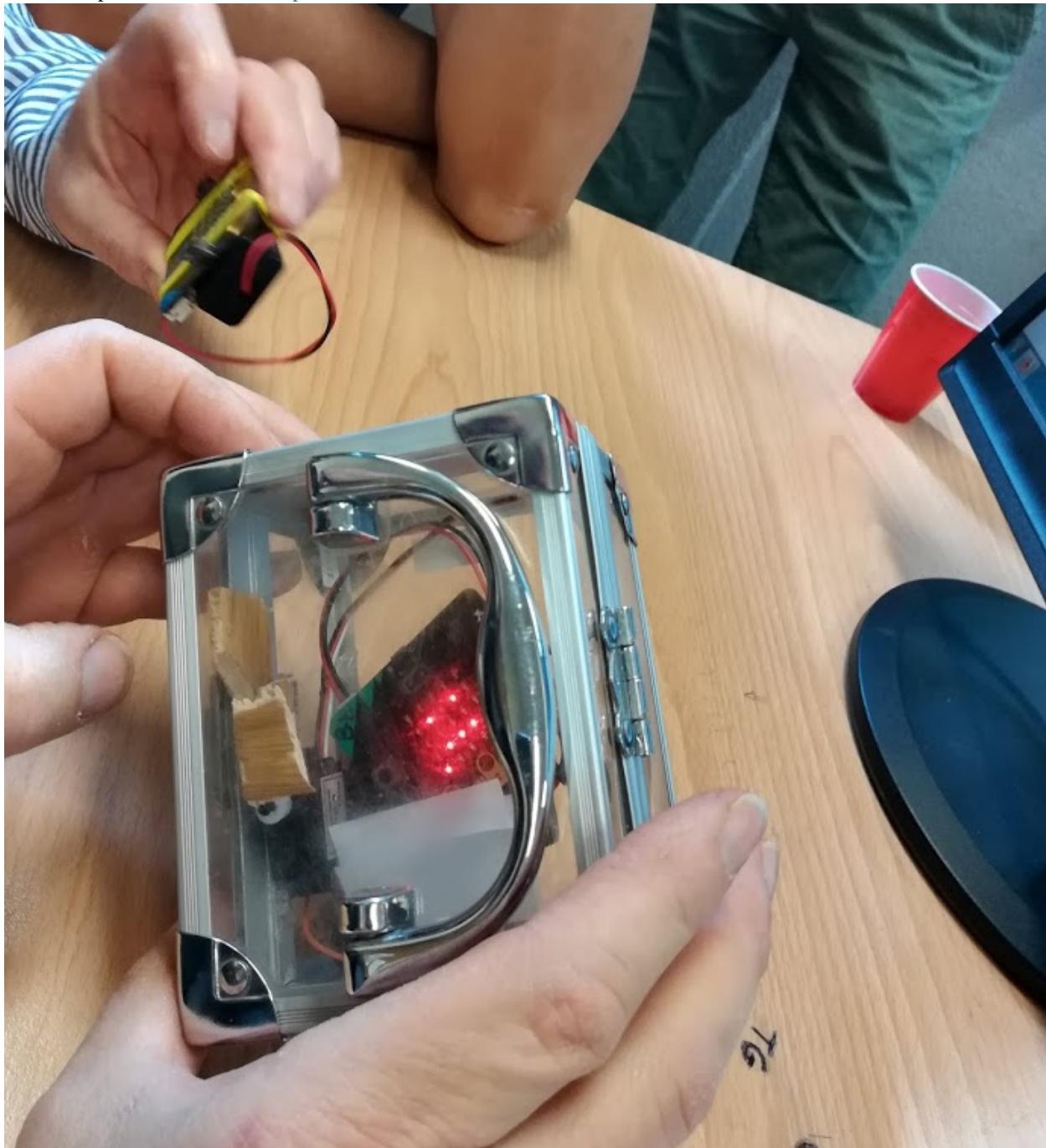
Exemple(s) d'utilisation

---

### Escape game

Nous avons utilisé le projet *Boîte fermée* pour un escape game proposé en stage.

— diaporama d'accueil : <http://url.univ-irem.fr/boite>



#### 2.1.2 Réalisation

## Fabriquer

Nous détaillons ici comment fabriquer et assembler le matériel nécessaire à la réalisation du projet *Boîte fermée*.

### Matériel :

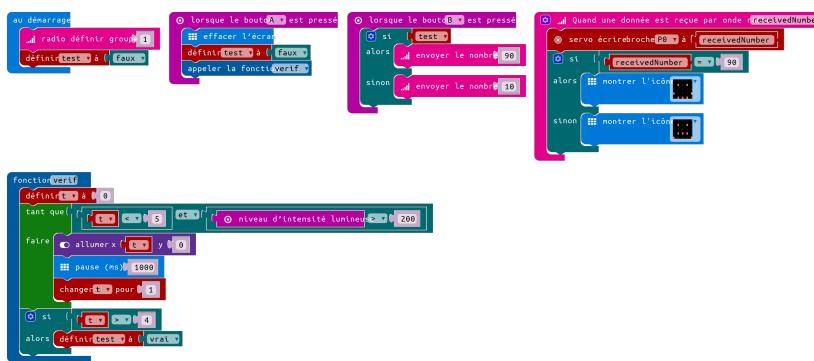
- 2 cartes microbit avec leur alimentation
- 1 servomoteur
- quelques fils de contact et pinces crocodiles
- une boite (transparente)
- pistolet à colle
- chutes de bois

### Assemblage :

- coller un morceau de bois d'environ 2 cm sur le servomoteur
- coller le servomoteur au niveau de l'ouverture de la boîte de façon à ce que la partie mobile se trouve au niveau du couvercle
- coller une cale de bois sur la partie inférieure de la boîte de façon à ce que la partie mobile du servo puisse vérrouiller lorsqu'elle vient en butée
- brancher l'alimentation sur servo sur les bornes 3v et GND du microbit et le fil de commande sur la borne P0

## Coder

Nous détaillons ici le code nécessaire à la réalisation du projet *Boîte fermée*.



## 2.2 Pierrot et Simon

### 2.2.1 Description

---

**À faire :** capture d'écran / gif animée

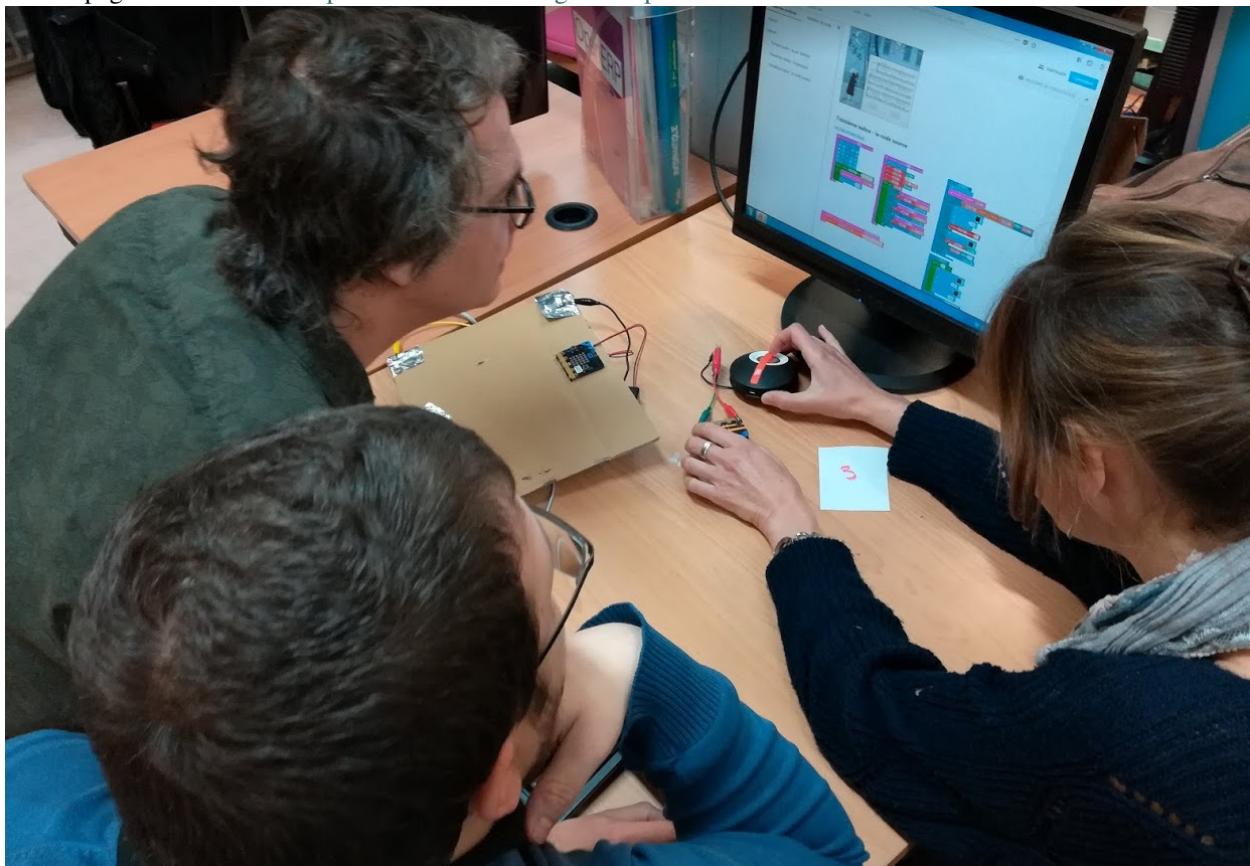
---

### Exemple(s) d'utilisation

#### Escape game

Nous avons utilisé le projet projetPierrot pour un escape game proposé en stage.

- diaporama d'accueil : <http://url.univ-irem.fr/pierrot>
- page de formation : <http://url.univ-irem.fr/algo1718-pierrot>



## 2.2.2 Réalisation

### Fabriquer

Nous détaillons ici comment fabriquer et assembler le matériel nécessaire à la réalisation du projet projetPierrot.

---

**À faire :** tout faire.

---

### Coder

Nous détaillons ici le code nécessaire à la réalisation du projet projetPierrot.

---

**À faire :** tout à faire !

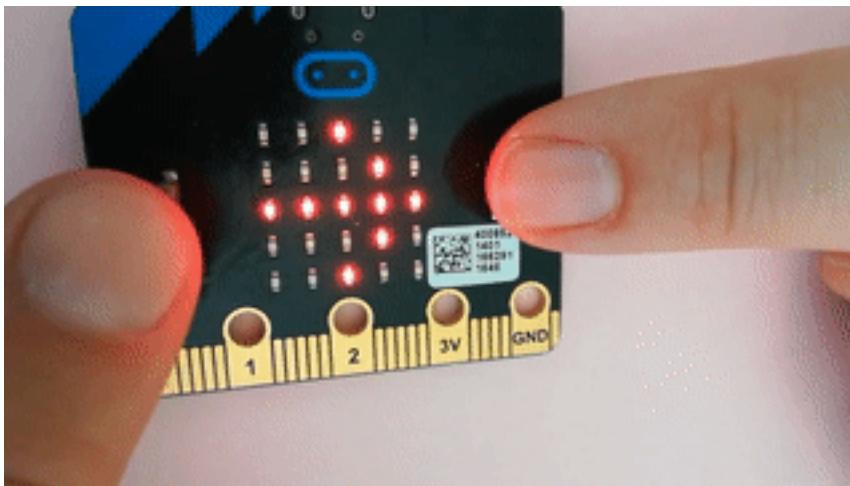
---

## 2.3 Go Fast

### 2.3.1 Description

L'objectif de ce projet est de créer un jeu : le **Go Fast**.

Ce jeu se joue à deux joueurs. Le gagnant est celui qui est le plus rapide à appuyer sur le bouton. Mais attention, si le bouton est déclenché trop tôt, alors la partie est perdue !



#### Règle du jeu

1. initialiser le Micro :bit
2. lancer une partie en pressant le bouton A ou le bouton B
3. un compte à rebours de 3 secondes s'affiche à l'écran
4. après un temps *indéterminé*, un symbole s'affiche à l'écran
5. **le gagnant** est le premier à appuyer sur son bouton
6. recommencer à l'étape 2. pour faire une nouvelle partie

#### Simulateur

**Avertissement :** L'image ci-dessous est dynamique. Une fois chargée (si tout se passe bien), elle vous propose une image qui simule le micro :bit.

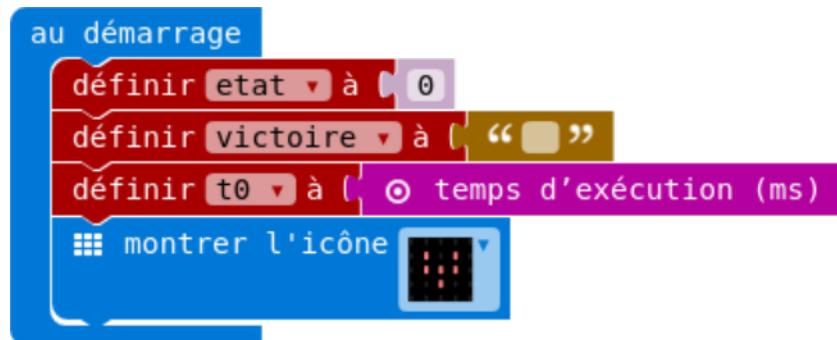
- cliquer sur le bouton A ou B pour lancer le jeu ;
- cliquer sur un des deux boutons pour désigner le gagnant ;
- cliquer (encore ?) sur un bouton pour relancer une partie.

### 2.3.2 Réalisation

#### Coder par blocs

#### Structure de base du programme

### Initialisation du programme



Quelques remarques concernant cette initialisation du micro :bit.

**Important :** Il y a dans ce jeu différents **états**. Le programme débute à l'état 0.

- etat = 0 : le jeu est en attente d'une partie. On quitte cet état en pressant le bouton A ou B.
- etat = 1 : le compte à rebours se lance. 3 secondes plus tard, on passe automatiquement à l'état 2.
- etat = 2 : la partie est en cours. Une pression sur le bouton A ou B permet de passer à l'état 3.
- etat = 3 : l'écran indique le bouton gagnant puis passe automatiquement à l'état 0.

---

#### Note :

##### — Victoire

La variable `victoire` permet d'afficher le gagnant. Ce sera un texte qui vaudra "A", "B" ou rien du tout "".

##### — Temps de référence

Pour déterminer les durées, nous allons utiliser et mettre à jour à la demande un temps de référence `t0`. Ainsi, la durée sera déterminée par la différence entre le temps d'exécution du micro :bit et ce temps de référence.

---

### Boucle principale

**Avertissement :** Lorsqu'un objet connecté est mis sous tension, le programme s'exécute après une brève phase d'initialisation.

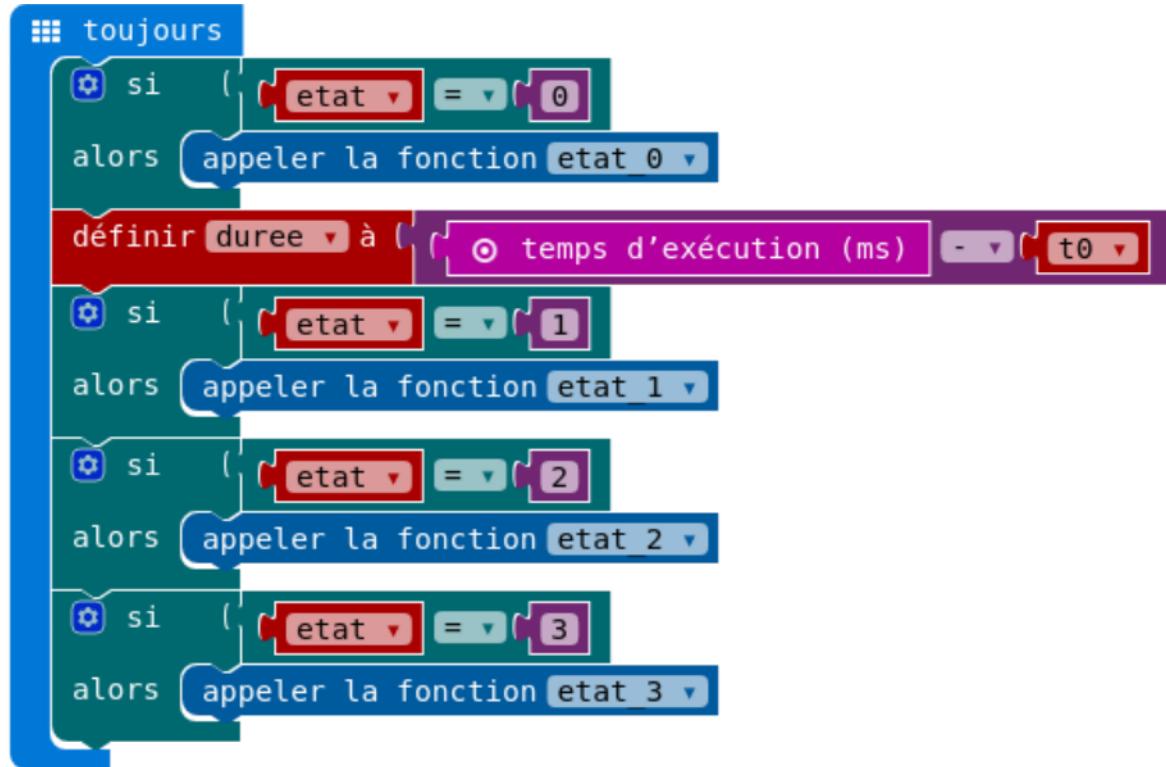
S'il n'y a pas de boucle, le programme ne s'exécute **qu'une seule fois** puis... plus rien !

En fait, un objet connecté est en permanence en train d'attendre que quelque chose se passe : un bouton pressé, un geste, une certaine température, luminosité ou autres.

Un peu comme votre clavier qui attend **en permanence** que vous appuyiez sur une touche.

C'est pour cela qu'il est nécessaire d'exécuter une boucle infinie qui met le microcontrôleur en état d'attente. Cette boucle infinie se nomme `toujours`.

Dans la boucle principale du programme, nous testerons donc les différents états. Pour chacun des états, nous appellerons la **fonction** dédiée.



**Astuce :** Pour que cette boucle n'exécute que la tâche souhaitée, nous testons la valeur de la variable `etat`.

Ainsi, à chaque itération de la boucle, tant que `etat` n'est pas modifiée, c'est toujours la même fonction qui est appelée.

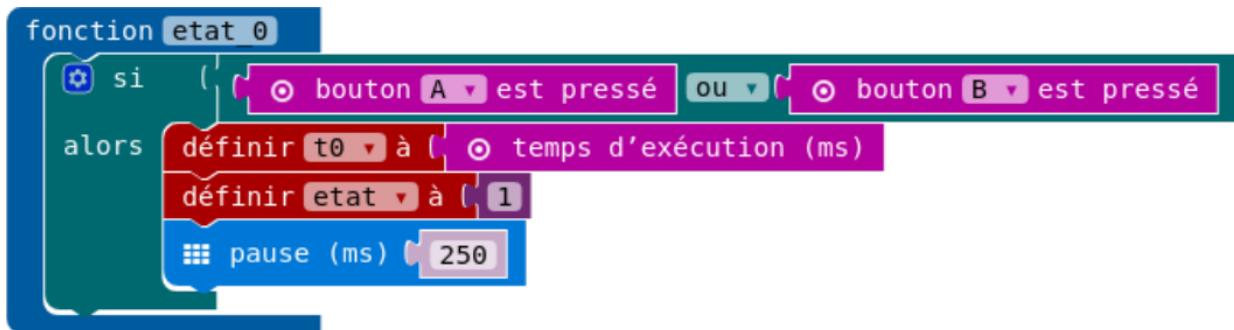
Cette méthode est très pratique pour afficher le compte à rebours (`état 1`) ou le gagnant (`état 3`).

**Note :** La durée d'exécution du jeu est mise à jour entre les tests des états 0 et 1.

Comme indiquée plus haut, la durée sera déterminée par la différence entre le temps d'exécution du micro:bit et le temps de référence  $t_0$ .

### La fonction `etat_0`

L'état 0 est un état de *repos* du micro:bit : il ne fait rien à part attendre qu'une partie démarre. Pour le sortir de cet état, il suffit de presser un des deux boutons.



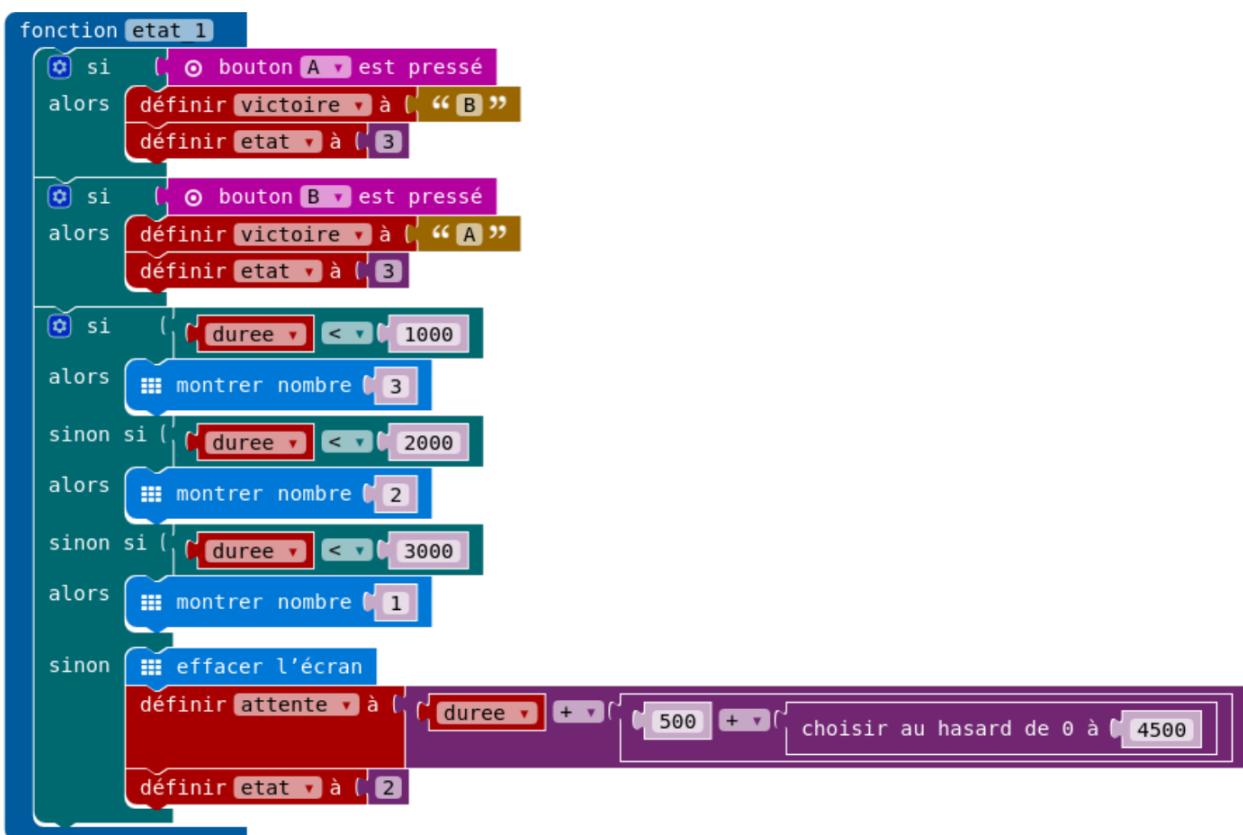
**Note :** Dès qu'un bouton est pressé, le temps de référence de cette nouvelle partie est réinitialisé.

Nous ajoutons une pause afin éviter de détecter un double appui ou un appui prolongé sur le bouton. Ceci entraînerait, inévitablement, la perte de la partie pour le joueur concerné.

---

### La fonction etat\_1

La fonction etat\_1 est dédiée à l'affichage du compte à rebours. La partie a débutée et si un joueur a le malheur de presser son bouton maintenant : la partie s'arrête et le nom du gagnant s'affiche en passant à l'état 3.



**Note :** Une fois que le compte à rebours est écoulé, on détermine de façon aléatoire un temps d'attente entre 500ms et 5000ms. On passe alors à l'état 2.

---

### La fonction etat\_2

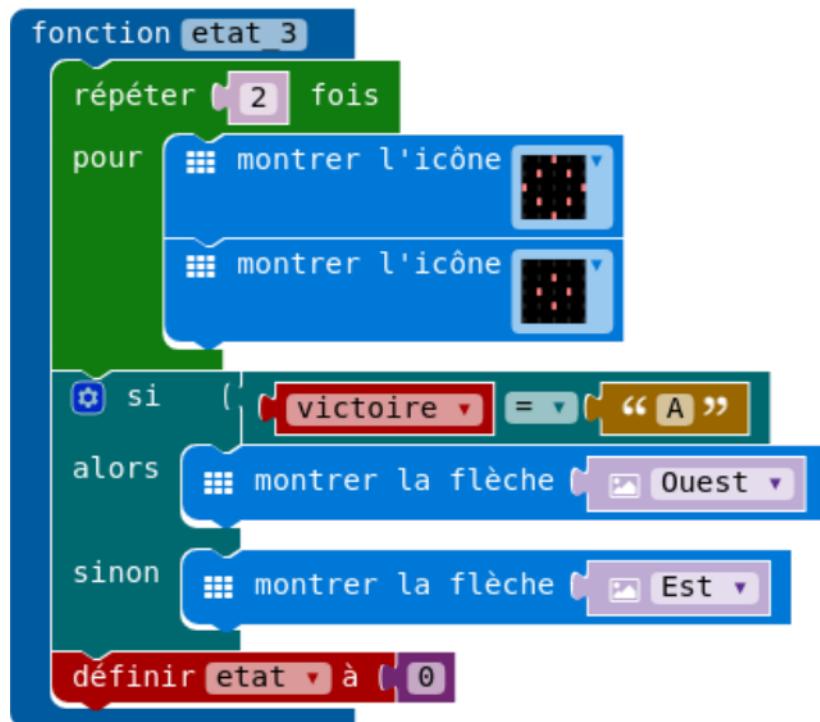
Cet état est le jeu à proprement parlé. Si un des joueurs appui sur son bouton avant que la première LED ne s'allume, la partie est immédiatement perdue pour lui.

En revanche, s'il est le premier à appuyer après la durée d'attente (aléatoire et déterminée à l'état 1), alors il est le gagnant.



### La fonction etat\_3

Cet état arrête la partie. Mais avant cela, il y a une petite animation, un léger suspens et puis une flèche indique le gagnant.



---

**Note :** Les flèches sont différencierées selon les points cardinaux. La flèche qui pointe vers la gauche est la flèche Ouest alors que celle pointant vers la droite est la flèche Est.

---

### Coder en Python

Le code nécessaire à la réalisation du projet *Go Fast* a été écrit en micropython. Vous trouverez ci-dessous :

- *Le code, étape par étape*
- *Le code final*

#### Le code, étape par étape

##### Initialisation du programme

1. Incluons la bibliothèque Micro :bit et la bibliothèque random pour générer de l'aléa :

```
from microbit import *
import random
```

2. Il y a dans ce jeu différents états. Initialement, le programme commence à l'état 0.

---

**Note :** L'utilisation de la variable `etat` permet de naviguer entre différent moment du programme.

---

- `etat = 0` : le jeu est en attente d'une partie. On quitte cet état en pressant le bouton A ou B.
- `etat = 1` : le compte à rebours se lance. 3 secondes plus tard, on passe automatiquement à l'état 2.
- `etat = 2` : la partie est en cours. Une pression sur le bouton A ou B permet de passer à l'état 3.
- `etat = 3` : l'écran indique le bouton gagnant puis passe automatiquement à l'état 0.

```
etat = 0
```

3. La variable `victoire` stocke le nom du bouton gagnant. Cette variable est un texte qui peut valoir "A" ou "B". En début de partie, nous l'initialisons à un texte vide :

```
victoire = ""
```

**Astuce :** Pour faire une pause d'une certaine durée, il n'est parfois pas recommandé d'utiliser la commande `sleep`.

En effet, `sleep` empêche l'exécution de toutes les autres directives du programme durant cette pause.

Il est préférable de déterminer une durée depuis un temps de référence et de tester, **à chaque boucle d'itération** la durée écoulée.

Pour cela, il faudra utiliser la commande `running_time()`.

4. Dans ce programme, il faut parvenir à déterminer une **durée**. Pour cela, nous allons utiliser la commande `running_time()` qui retourne la durée depuis laquelle le Micro :bit est sous tension.

Après avoir initialisé notre temps de référence `t0` (au démarrage du micro :bit ou en appuyant sur un bouton), nous calculerons la différence `running_time() - t0` qui déterminera la durée.

Initialisons `t0`

```
t0 = running_time()
```

5. Pour déterminer une durée, définissons la fonction `chrono()`.

La durée écoulée entre le temps de référence `t0` et le moment de l'appel de la fonction.

```
def chrono(t0):
    t1 = running_time()
    duree = t1-t0
    return duree
```

6. Enfin, affichons à l'écran une image pour indiquer le programme est en attente.

```
display.show(Image.HEART_SMALL)
```

## Boucle

**Avertissement :** Lorsqu'un objet connecté est mis sous tension, le programme s'exécute après une brève phase d'initialisation.

S'il n'y a pas de boucle, le programme ne s'exécute **qu'une seule fois** puis... plus rien !

En fait, un objet connecté est en permanence en train d'attendre que quelque chose se passe : un bouton pressé, un geste, une certaine température, luminosité ou autres.

Un peu comme votre clavier qui attend **en permanence** que vous appuyiez sur une touche.

C'est pour cela qu'il est nécessaire d'exécuter une boucle infinie qui met le microcontrôleur en état d'attente.

- La phase de configuration est terminée. Passons maintenant à la boucle qui... tourne en boucle.

```
while True:
```

2. Configurons l'état 0 du programme qui attend que l'un des joueurs presse son bouton.

Lorsqu'un bouton est pressé, il faut initialiser le temps de référence `t0` puis changer d'état. La pause `sleep(250)` permet d'éviter un *double-clic* du bouton qui entraînerait la fin prématurée de la partie.

```
if etat == 0:  
    if button_a.is_pressed() or button_b.is_pressed():  
        # remise à zéro de la durée de la partie  
        t0 = running_time()  
        etat = 1  
        sleep(250)
```

3. Il est temps de relever la durée écoulée depuis le temps de référence.

```
duree = chrono(t0)
```

4. L'état 1 affiche le compte à rebours.

```
if etat == 1:
```

Mais attention, si un joueur presse un bouton dans cet état, la partie est immédiatement perdue (et on passe alors à l'état 3 sans passer par la case *Départ*).

```
if button_a.is_pressed():  
    victoire = "B"  
    etat = 3  
if button_b.is_pressed():  
    victoire = "A"  
    etat = 3
```

Affichons le compte à rebours (sans oublier que les durées sont en millisecondes).

```
if duree < 1000:  
    display.show("3")  
elif duree < 2000:  
    display.show("2")  
elif duree < 3000:  
    display.show("1")  
else:
```

Puis après les 3 secondes, on va effacer l'écran, déterminer une durée d'attente au hasard, choisir une image au hasard et enfin passer à l'état 2.

```
else:  
    display.clear()  
    attente = duree + random.randint(500, 5000)  
    imageAuHasard = random.choice(Image.ALL_CLOCKS)  
    etat = 2
```

5. L'état 2 est le jeu à proprement dit. Le plus rapide des joueur à presser son bouton sera enfin déclaré vainqueur.

```
if etat == 2:
```

Mais attention, si un joueur clique avant la durée d'attente alors son adversaire gagne immédiatement (et on passe alors à l'état 3).

```
if duree < attente:  
    if button_a.is_pressed():  
        victoire = "B"  
        etat = 3
```

(suite sur la page suivante)

(suite de la page précédente)

```
if button_b.is_pressed():
    victoire = "A"
    etat = 3
```

Une fois que la durée a dépassée l'attente souhaitée, alors là le plus rapide gagne.

```
if duree > attente:
    display.show(imageAuHasard)
if button_a.is_pressed():
    victoire = "A"
    etat = 3
if button_b.is_pressed():
    victoire = "B"
    etat = 3
```

6. L'état 3 permet d'indiquer le vainqueur.

```
if etat == 3:
```

D'abord une petite animation pour le suspens.

```
for i in range(5):
    display.show(Image.HEART_SMALL)
    sleep(100)
    display.show(Image.HEART)
    sleep(100)
```

Puis enfin le vainqueur. Nous allons utiliser une flèche pointant vers le bouton en question. La flèche Ouest qui pointe le bouton de gauche (*A*) et la flèche Est qui pointe le bouton *B* de droite.

```
if victoire == "A":
    display.show(Image.ARROW_W)
if victoire == "B":
    display.show(Image.ARROW_E)
etat = 0
```

Pour finir, une pause qui ne fait pas de mal (peut être même que ça économise les piles...).

```
sleep(100)
```

## Le code final

```
1 # -*- coding: utf-8-*# Encoding cookie added by Mu Editor
2 # code utilisé pour l'atelier colloque C2i collège
3 # auteur : groupe InEFLP IREM de Marseille
4
5 from microbit import *
6 import random
7
8 # différente ETAT du jeu
9 # si etat == 0 : attente du début de partie (A ou B)
10 # si etat == 1 : compte à rebours
11 # si etat == 2 : en cours de partie
12 # si etat == 3 : affichage du gagnant
13 etat = 0
14
```

(suite sur la page suivante)

(suite de la page précédente)

```

15 # victoire == "A" : bouton A gagne
16 # victoire == "B" : bouton B gagne
17 # victoire == "" : personne
18 victoire = ""

19
20 # fonction CHRONO qui sert de chronomètre
21 # retourne la durée écoulée depuis
22 # l'instant t0
23 t0 = running_time()

24
25 def chrono(t0):
26     t1 = running_time()
27     duree = t1-t0
28     return duree

29
30 display.show(Image.HEART_SMALL)

31
32 while True:

33
34     if etat == 0:
35         if button_a.is_pressed() or button_b.is_pressed():
36             # remise à zéro de la durée de la partie
37             t0 = running_time()
38             etat = 1
39             sleep(250)

40
41     # duree de la partie
42     duree = chrono(t0)

43
44     if etat == 1:
45         # si les joueurs sont trop pressés, ils perdent...
46         if button_a.is_pressed():
47             victoire = "B"
48             etat = 3
49         if button_b.is_pressed():
50             victoire = "A"
51             etat = 3

52
53         # affichage du compte à rebours
54         if duree < 1000:
55             display.show("3")
56         elif duree < 2000:
57             display.show("2")
58         elif duree < 3000:
59             display.show("1")
60         else:
61             # initialisation d'une partie
62             #
63             # effacer écran
64             display.clear()
65             # attente et image aléatoire
66             attente = duree + random.randint(500, 5000)
67             imageAuHasard = random.choice(Image.ALL_CLOCKS)
68             etat = 2

69
70     if etat == 2:
71         # si les joueurs sont trop pressés, ils perdent...

```

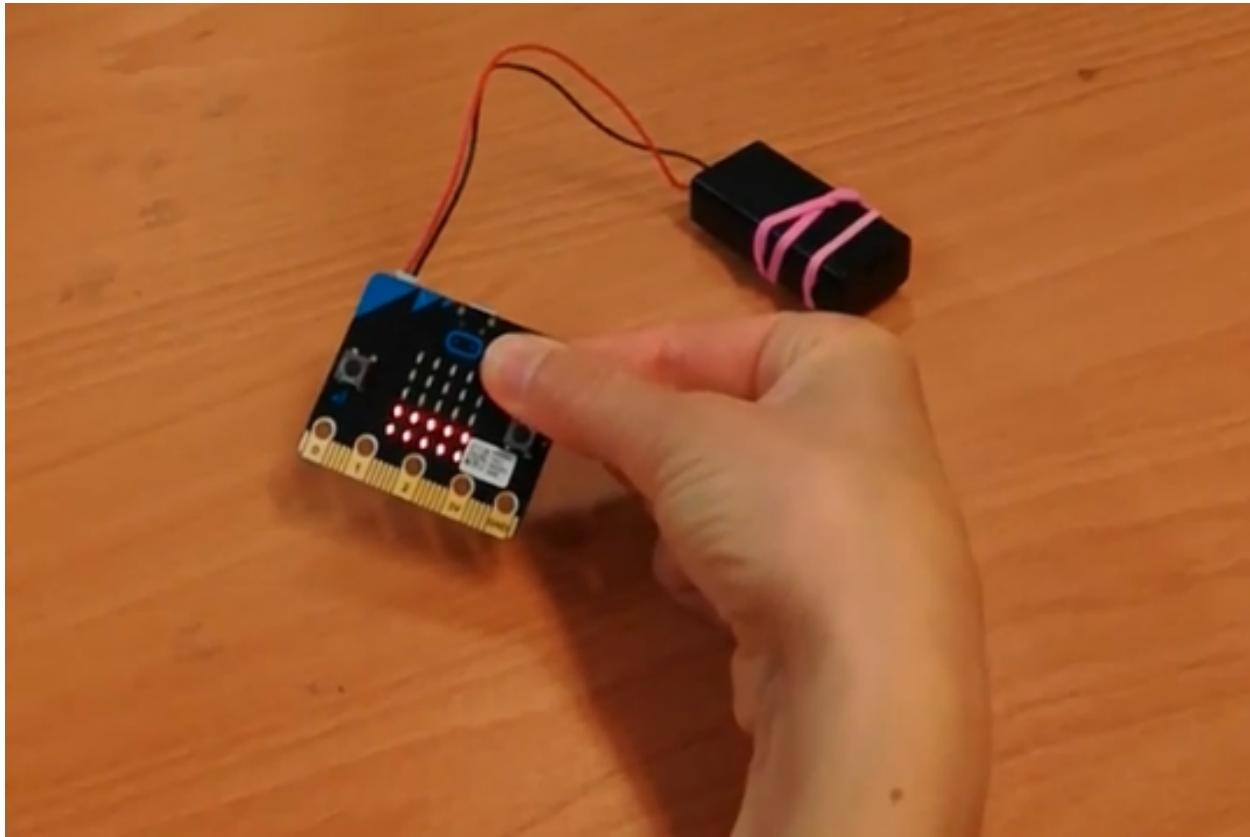
(suite sur la page suivante)

(suite de la page précédente)

```
72     if duree < attente:
73         if button_a.is_pressed():
74             victoire = "B"
75             etat = 3
76         if button_b.is_pressed():
77             victoire = "A"
78             etat = 3
79
80         # détermination du plus rapide des 2 joueurs
81         if duree > attente:
82             display.show(imageAuHasard)
83             if button_a.is_pressed():
84                 victoire = "A"
85                 etat = 3
86             if button_b.is_pressed():
87                 victoire = "B"
88                 etat = 3
89
90         if etat == 3:
91             # petite animation pour le suspens
92             for i in range(5):
93                 display.show(Image.HEART_SMALL)
94                 sleep(100)
95                 display.show(Image.HEART)
96                 sleep(100)
97             # affichage du gagnant
98             if victoire == "A":
99                 display.show(Image.ARROW_W)
100            if victoire == "B":
101                display.show(Image.ARROW_E)
102            etat = 0
103
104 sleep(100)
```

## 2.4 Températures

### 2.4.1 Description



Le code téléchargé dans le Micro:bit permet d'afficher un **code secret**. Pour cela, il faut que la température augmente et dépasse les 34°C. L'écran affiche une jauge qui se remplit.

Une fois la température atteinte, le code secret s'affiche après une petite animation.

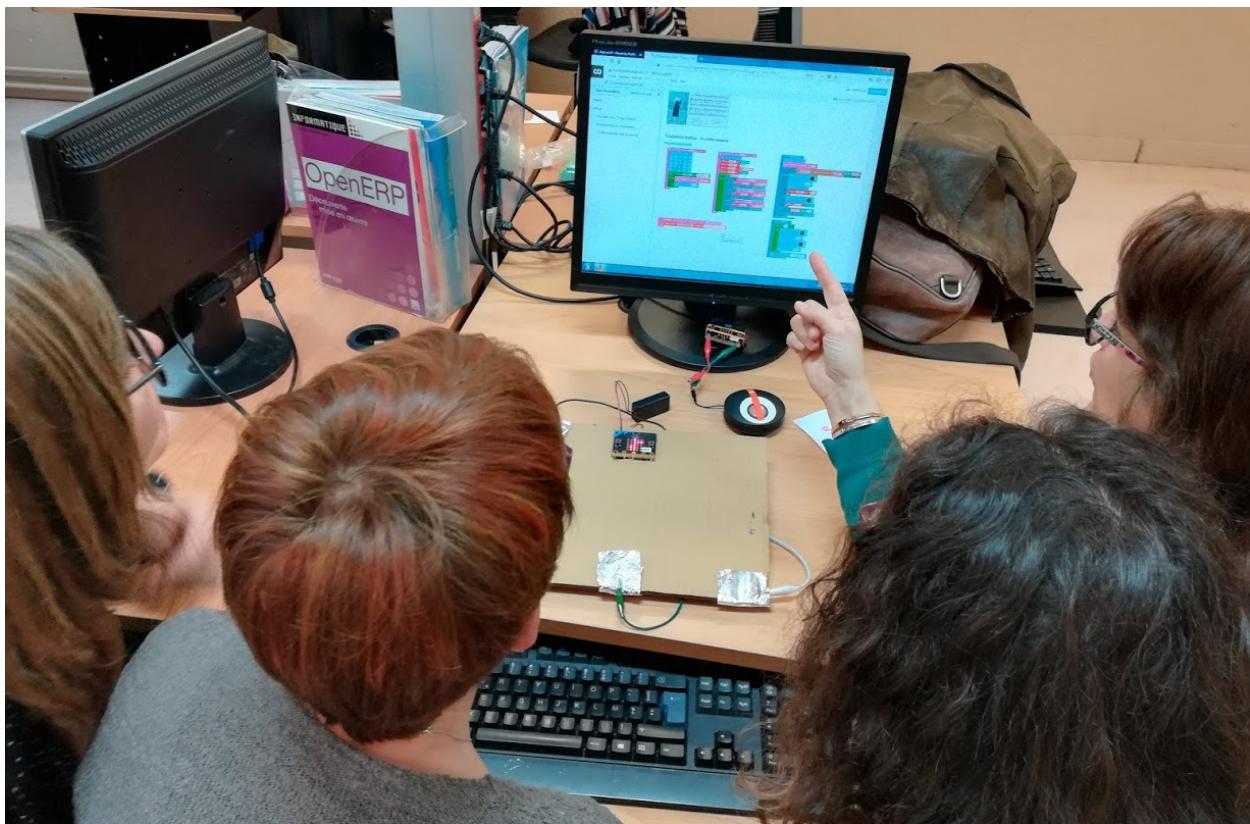
#### Exemple(s) d'utilisation

##### Escape game

Nous avons utilisé le projet `projetTemp` pour un escape game proposé en stage.

- diaporama d'accueil : <http://url.univ-irem.fr/temp>
- page de formation : <http://url.univ-irem.fr/algo1718-temp>

Le but était d'afficher un code secret permettant d'accéder à l'énigme suivante.



Les stagiaires devaient donc prendre connaissance du diaporama d'accueil qui les renvoyaient vers des indices et le code source téléchargé dans le micro:bit.

À l'aide de l'étude du code source, des indices et la bidouille, les stagiaires devaient effectuer les manipulations nécessaires à l'affichage du code source.

De très bons moments pour tous !

## 2.4.2 Réalisation

### Fabriquer

Pour le projet projetTemp, il n'y a besoin de presque rien :

- carte micro:bit;
- alimentation électrique (pile par exemple).

### Coder

Le code nécessaire à la réalisation du projet projetTemp a été écrit en micropython. Vous trouverez ci-dessous :

- [Le code, étape par étape](#)
- [Le code final](#)

### Le code, étape par étape

1. Incluons la bibliothèque Micro:bit

```
from microbit import *
```

2. Créons les images qui nous servirons à animer l'écran. La luminosité d'une diode varie de 0 (éteinte) à 9 (maximale).

Lorsque la température augmente, l'affichage passe progressivement de image1 à image 5.

```
image1 = Image(  
    '00000:'  
    '00000:'  
    '00000:'  
    '00000:'  
    '99999')  
image2 = Image(  
    '00000:'  
    '00000:'  
    '00000:'  
    '99999:'  
    '77777')  
image3 = Image(  
    '00000:'  
    '00000:'  
    '99999:'  
    '77777:'  
    '77777')  
image4 = Image(  
    '00000:'  
    '99999:'  
    '77777:'  
    '77777:'  
    '77777')  
image5 = Image(  
    '99999:'  
    '77777:'  
    '77777:'  
    '77777:'  
    '77777')
```

3. Il y aura deux états dans le jeu :

- La variable victoire est vrai et l'écran affiche le code secret.
- la variable victoire est fausse et l'écran affiche l'énigme.

Au début, la variable est donc fausse.

```
victoire = False
```

4. La phase de configuration est terminée. Passons maintenant à la boucle qui... tourne en boucle.

Tout ce qui suivra cette codee sera donc indenté (tabulation).

```
while True:
```

5. Nous envisageons trois actions possibles :

- (a) le jeu se réinitialise grâce au bouton A ;

```
if button_a.is_pressed():  
    victoire = False
```

- b) le jeu est gagné et l'écran affiche le code final (après une petite animation) ;

```

if victoire:
    # petite image joyeuse
    display.show(Image.HAPPY)
    sleep(500)
    # code secret à afficher...
    display.scroll("XXXXXX")

```

(c) le jeu est en cours et l'écran affiche les images.

```
if not victoire:
```

Lire la température

```
temp = temperature()
```

Plus la température augmente, plus les images affichées remplissent l'écran

```

if temp < 29:
    display.clear()
elif 29 <= temp < 30:
    display.show(image1)
elif 30 <= temp < 31:
    display.show(image2)
elif 31 <= temp < 32:
    display.show(image3)
elif 32 <= temp < 33:
    display.show(image4)
elif 33 <= temp < 34:
    display.show(image5)

```

et enfin, si la température dépasse 34°C, alors là on passe en mode victoire vrai. On ajoute une petite animation pour montrer que la victoire approche.

```

elif 34 <= temp:
    victoire = True
    # petite animation
    for i in range(2):
        display.show(Image.SQUARE_SMALL)
        sleep(100)
        display.show(Image.SQUARE)
        sleep(100)

```

Pour finir, une pause syndicale de 500ms.

```
sleep(500)
```

## Le code final

```

1  # -*- coding: utf-8-*# Encoding cookie added by Mu Editor
2  from microbit import *
3
4  # définir mes images perso
5  # pour les lignes qui se colorent
6  image1 = Image(
7      '00000:'
8      '00000:'

```

(suite sur la page suivante)

(suite de la page précédente)

```
9      '00000:'  
10     '00000:'  
11     '99999')  
12 image2 = Image(  
13     '00000:'  
14     '00000:'  
15     '00000:'  
16     '99999:'  
17     '77777')  
18 image3 = Image(  
19     '00000:'  
20     '00000:'  
21     '99999:'  
22     '77777:'  
23     '77777')  
24 image4 = Image(  
25     '00000:'  
26     '99999:'  
27     '77777:'  
28     '77777:'  
29     '77777')  
30 image5 = Image(  
31     '99999:'  
32     '77777:'  
33     '77777:'  
34     '77777:'  
35     '77777')  
36  
37 # booléen pour savoir si l'énigme est réussie  
38 victoire = False  
39  
40 # à faire toujours et toujours...  
41 while True:  
42     # utiliser le bouton A pour réinitialiser  
43     if button_a.is_pressed():  
44         victoire = False  
45  
46     # si l'énigme est résolue  
47     if victoire:  
48         # petite image joyeuse  
49         display.show(Image.HAPPY)  
50         sleep(500)  
51         # code secret à afficher...  
52         display.scroll("XXXXXX")  
53  
54     # si l'énigme n'a pas été résolue  
55     if not victoire:  
56         # lire la température (en °C)  
57         temp = temperature()  
58         # affichage des images en fonction  
         # de temp  
59         if temp < 29:  
60             display.clear()  
61         elif 29 <= temp < 30:  
62             display.show(image1)  
63         elif 30 <= temp < 31:  
64             display.show(image2)
```

(suite sur la page suivante)

(suite de la page précédente)

```
66     elif 31 <= temp < 32:
67         display.show(image3)
68     elif 32 <= temp < 33:
69         display.show(image4)
70     elif 33 <= temp < 34:
71         display.show(image5)
72     # victoire !
73     elif 34 <= temp:
74         victoire = True
75         # petite animation
76         for i in range(2):
77             display.show(Image.SQUARE_SMALL)
78             sleep(100)
79             display.show(Image.SQUARE)
80             sleep(100)
81         sleep(500)
```

## 2.5 Coffre fort

### 2.5.1 Description

---

À faire : capture d'écran / gif animée

---

#### Exemple(s) d'utilisation

##### Escape game

Nous avons utilisé le projet *Coffre fort* pour un escape game proposé en stage.

- diaporama d'accueil : <http://url.univ-irem.fr/coffre>
- page de formation : <http://url.univ-irem.fr/algo1718-coffre>



## 2.5.2 Réalisation

### Fabriquer

Nous détaillons ici comment fabriquer et assembler le matériel nécessaire à la réalisation du projet *Coffre fort*.

---

**À faire :** tout faire.

---

### Coder

Nous détaillons ici le code nécessaire à la réalisation du projet *Coffre fort*.

---

**À faire :** tout à faire !

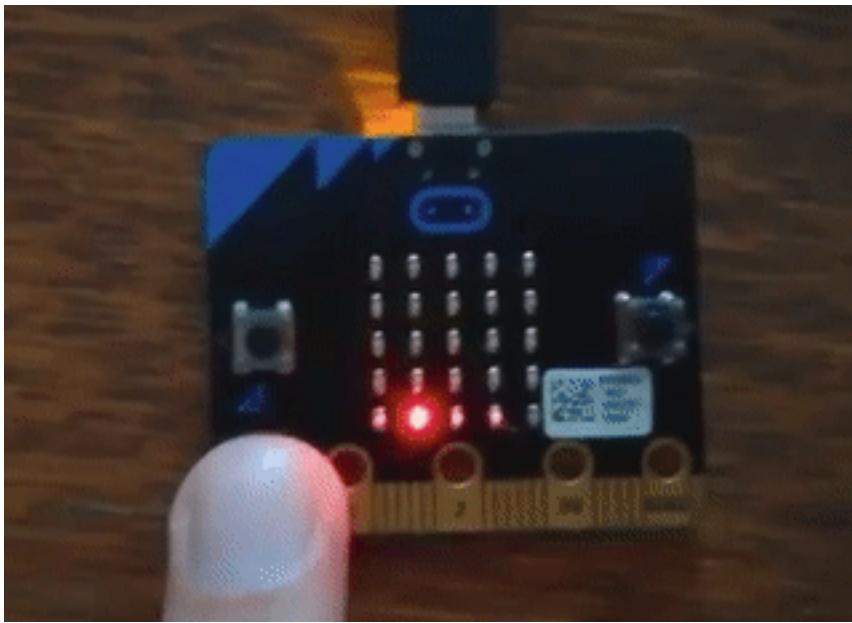
---

## 2.6 Planche de Galton

### 2.6.1 Description

Le but de ce programme est de simuler le parcours de billes sur une planche de Galton. Lorsque la bille tombe, elle a deux possibilités : elle descend verticalement ou elle descend verticalement en se décalant horizontalement vers la droite.

La bille position de la bille est représentée par l'allumage d'une diode.



#### Type de programmation :

Le projet est ici programmé en python.

#### Exemple(s) d'utilisation

- Domaine statistiques et probabilités du programme de mathématiques de Bac Pro
- Accompagnement personnalisé pour des élèves de seconde Bac Pro

### 2.6.2 Réalisation

#### Coder

Nous détaillons ici le code nécessaire à la réalisation du projet *Planche de Galton*.

```
from microbit import *
from random import random, seed

n = [0, 0, 0, 0, 0]      # le tableau contenant les compteurs
```

(suite sur la page suivante)

(suite de la page précédente)

```
def aff(n, m):          # la fonction affichant le graph
    q = n // 9           # nombre de led éclaire totalement
    r = n % 9             # portion de la dernière led éclaire
    for i in range(0, q):
        display.set_pixel(m, 4-i, 9)
    display.set_pixel(m, 4-q, r)

def chute(t):            # fonction affichant la chute
    display.clear()
    y, x = 0, 0
    display.set_pixel(x, y, 9)
    sleep(t)
    while y < 4:
        display.clear()
        if random.randint(0, 1):      # si aléa entre 0 ou 1 est vrai
            y = y + 1                # on augmente y de 1
        else:
            x = x + 1
            y = y + 1
        display.set_pixel(x, y, 9)
        sleep(t)
    n[x] = n[x]+1            # incrementation du compteur de la position x
    display.set_pixel(x, y, 1)

while True:
    if button_a.is_pressed():
        chute(500)

    elif button_b.get_presses():
        n = [0, 0, 0, 0, 0]
        for k in range(80):
            chute(round(500 / (1.05**k))) # accélération de la chute
            for j in range(5):
                aff(n[j], j)
            sleep(200)
        print(n)
```

### Fabriquer

Nous détaillons ici comment fabriquer et assembler le matériel nécessaire à la réalisation du projet *Planche de Galton*.

---

**À faire :** tout faire.

---

# CHAPITRE 3

---

## Index et page de recherche

---

- genindex
  - search
-



---

## Index

---

### A

accéléromètre, 56  
audio, 38

### B

bouton, 39, 44

### D

durée, 39, 44

### G

galton, 57

### L

luminosité, 37

### M

micropython, 38, 44, 51, 56, 57

### P

programmation par blocs, 37, 39

### S

servo, 37

### T

température, 51