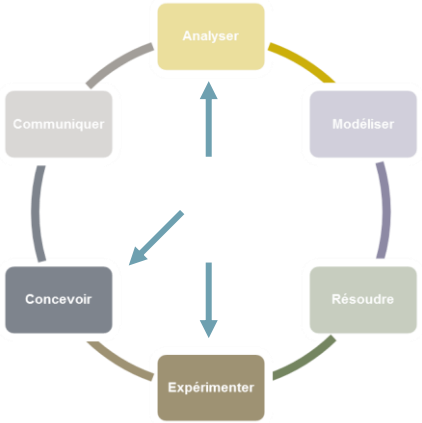
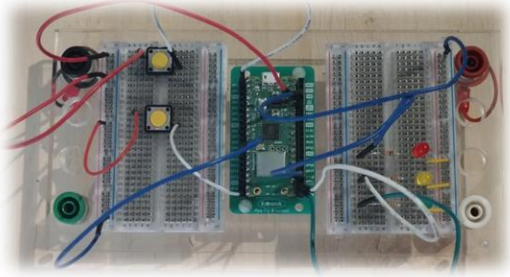
TP CPGE S2



INITIATION À MICROPYTHON 1 : COMMANDE FEUX TRICOLORES

- Concevoir : Modifier un programme et l'implanter dans un système cible
- Expérimenter : Etre capable d'effectuer un montage électronique simple



S. JUHEL
J. Ferry, Versailles

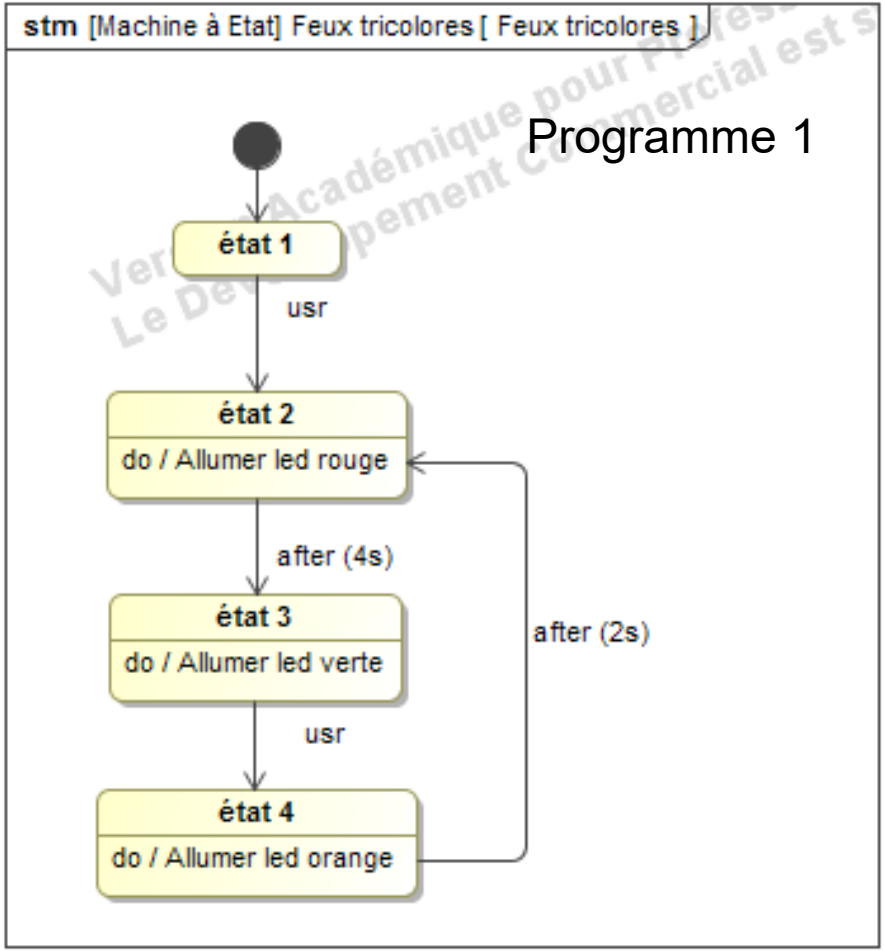
Appuyer sur Ctrl-L pour mettre en plein écran.
Appuyer sur ⊞ + tab pour basculer d'un document à un autre.

Diagramme d'états

L'objectif est de réaliser la commande d'un feu tricolore à l'aide d'une carte associée à un microcontrôleur en respectant ce diagramme d'états.

L'écriture du programme python suivra les règles explicitées dans ce sujet.

Lorsque le système est au repos (état1), tout est arrêté.

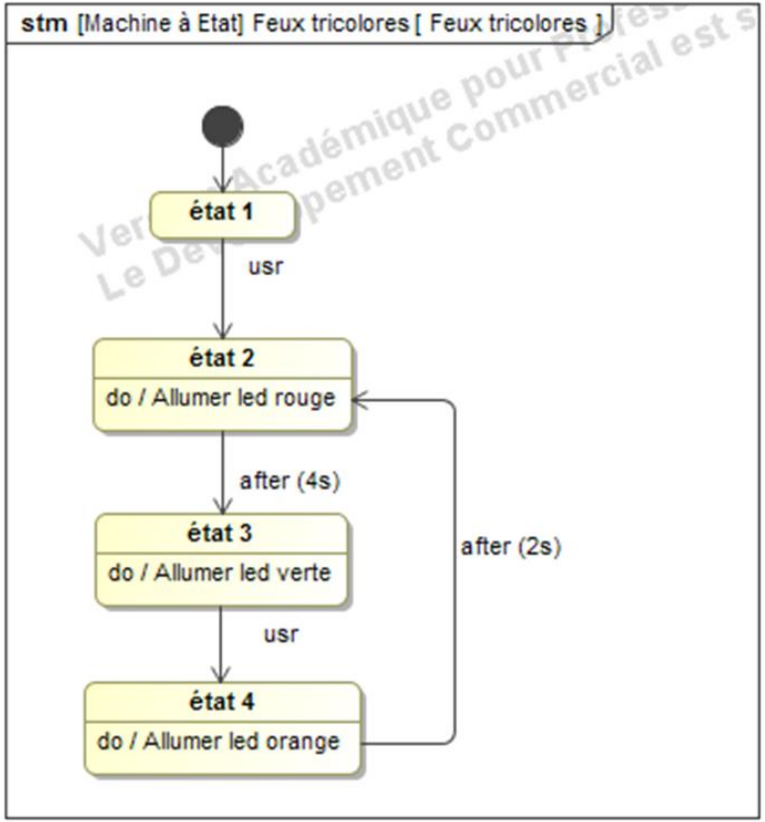


L'interprétation du diagramme est sur la page suivante.

Cahier des charges

- Le cahier des charges est le suivant :
  - Une première impulsion sur le bouton usr et la led rouge s'allume ;
  - Au bout de 4s, elle s'éteint et la led verte s'allume.
  - Si usr est appuyé de nouveau alors la led verte s'éteint et la led orange s'allume pendant 2s, puis la rouge.
  - Et ainsi de suite…

  - Ce système qui réagi à l'apparition d'évènements (comme appuyer sur usr) est appelé **Système à Evènements Discrets**. (SED).
  - On vous propose de programmer la Raspberry Pi Pico à l'aide du diagramme de ce diagramme (dit d'états).



Pico microprocesseur RP2040

La carte Pico contient un microcontrôleur RP2040 (133 MHz) et propose :

- 26 entrées/sorties (ports) binaires (GPIO*) et pouvant fournir un signal modulé à largeur d'impulsion (PWM – Pulse Width Modulation) utilisées pour la commande du hacheur et la variation de vitesse d'un moteur à courant continu par exemple (voir TP ventilo). On choisit en début de programme si tel port est une entrée ou une sortie. La technologie utilisée est CMOS (3,3 V).
- 3 entrées analogiques converties sur 12 bits.
- Des ports pouvant être configurés pour être des liaisons séries, bus I²C ou SPI non utilisés dans cette séance.

* GPIO : General Purpose Input Output

Raspberry Pi Pico Wifi

Led verte test

Port micro USB : utile pour la communication et l'alimentation (5V)

Attention FRAGILE
Ne pas mettre ses doigts dessus
Jamais plus de 3,3 V en entrée

Bouton BOOTSEL pour mise à niveau.
Ne pas toucher

3V3 (OUT) : sortie Alimentation 3,3 V pour alimenter des périphériques (technologie CMOS)

Microcontrôleur RP2040

Quartz

Module WIFI en option

Masse GrouND (GND)

26 Entrées/sorties binaires GPIO (General Purpose Input Output)

GP26, 27 et 28 sont équipées d'un convertisseur analogique/numérique 12 bits, 3,3 V max

Pour en savoir plus sur les entrées/sorties fig. pico

Inventaire

- Carte Pico et un câble micro USB
- Plaque de prototypage rapide (breadboard)
- 3 leds de couleurs verte, orange (ou jaune) et rouge
- 3 résistances de 470 Ω environ
- Des fils…
- 4 boutons poussoir (déjà placés sur une des plaques breadboard)

Raspberry Pi Pico et micropython

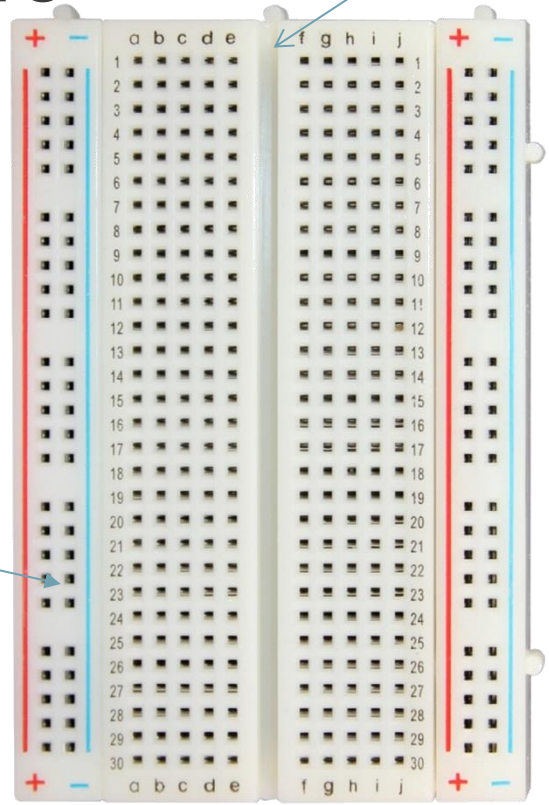
Si débutant en micropython, lisez le début de **l'aide micropython fournie** et lisez (et effectuez) ses **3** premières fiches, pour prendre en main cette carte et Thonny. (Thonny.exe est disponible sur C:)

Si pb de connexion appeler le professeur.

N'oubliez pas de sauvegarder régulièrement votre programme. Le prochain TP sera la suite de celui-ci.

Plaquette montage sans soudure

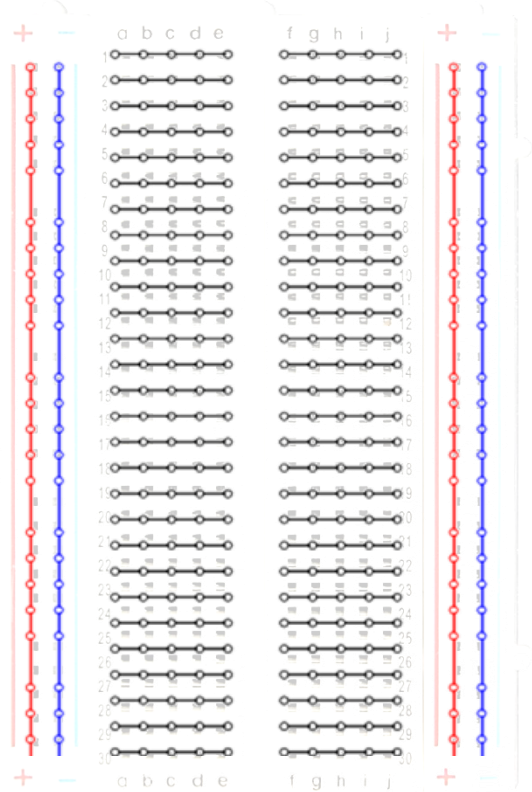
La plaquette permet de relier les composants sans soudure pour des tests.
Les composants à multiples entrées (comme le hacheur) doivent être placés à cheval sur la colonne centrale.
La colonne + (rouge) sera branchée au 3.3V de la carte, et la colonne – avec GND (ground).



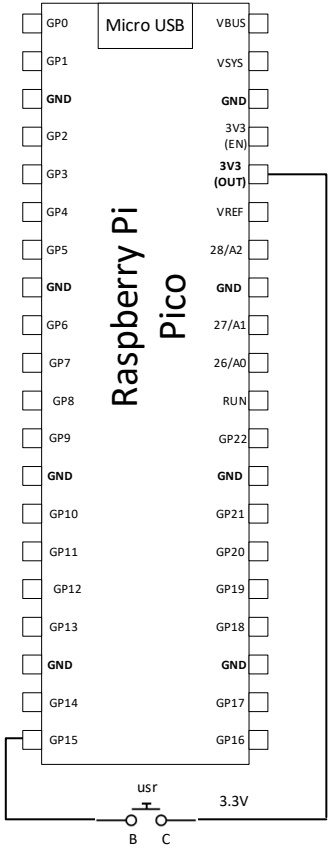
Cliquer sur l'image pour comprendre les connections

Plaquette montage sans soudure

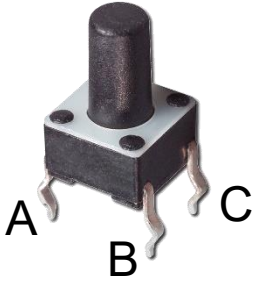
La plaquette permet de relier les composants sans soudure pour des tests.
Les composants à multiples entrées (comme le hacheur) doivent être placés à cheval sur la colonne centrale.
La colonne + (rouge) sera branchée au 3.3V de la carte, et la colonne – avec GND (ground).



Câblage pour le programme 1



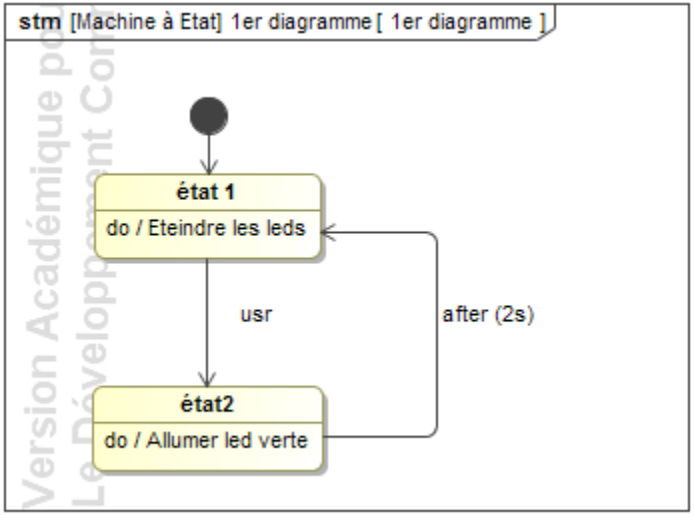
Le bouton fourni est normalement ouvert. Donc par défaut aucun courant ne passe.
Bien veiller à placer le bouton à cheval sur la colonne centrale qui sépare électriquement la plaque (breadboard) en 2.
Comme on place un fil relié au 3,3V de la carte sur un des connecteurs du bouton, cela signifie que lorsque vous appuierez dessus, le signal électrique 3,3V entrera dans la carte par le GP15 (voir câblage proposé). Il sera interprété comme un 1 logique par la carte.
Pour information, cette entrée sera compris par le futur programme python car déclarée comme suis : `usr=Pin(15,Pin.IN,Pin.PULL_DOWN)`



IMPORTANT, câblage bouton poussoir : A et B sont toujours liés. (tout comme C et D). Lorsque vous appuyez sur le bouton, vous connectez (A,B) à (C,D).

Programme d'introduction 1/5

On va apprendre à programmer en python la Pico, en suivant l'algorithme représenté par le diagramme d'états ci-après.



Le programme fourni comporte 6 parties qui s'écrivent à la suite :
1 importation des bibliothèques utiles,
2 déclaration des entrées, interfaces entre la carte et le monde extérieur (ici le bouton usr à câbler voir page précédente),
3 déclaration des sorties (ici la led verte interne)
4 déclaration et initialisation des constantes du programme,
5 écriture des fonctions utiles et des interruptions incluant l'évolution du graphe lié aux évènements externes,
6 écriture du programme principal (boucle while True:) (incluant l'évolution du graphe ainsi que l'affectation des sorties).

- Au démarrage, l'état1 est actif. (représenté par le point noir qui pointe sur l'état1) : les leds doivent être éteintes.
- si on appuie sur le bouton usr, l'état1 est désactivé et l'état2 est activé : on doit allumer la led verte ;
- Au bout de 2s, l'état2 se désactive et l'état1 s'active.
- Et ainsi de suite…

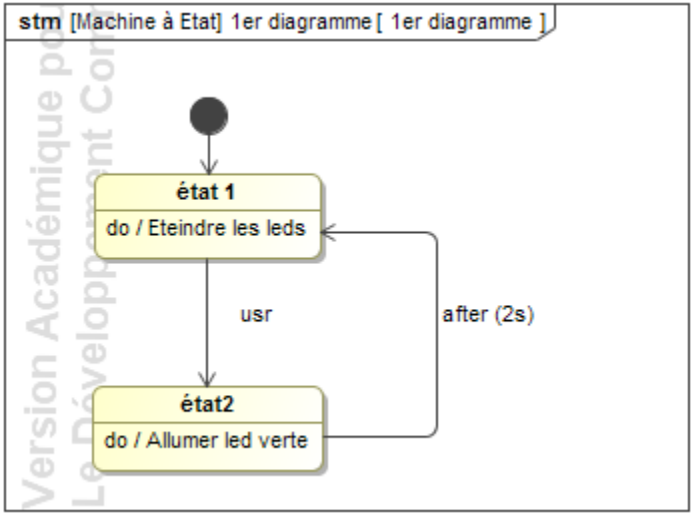
Programme d'introduction 2/5

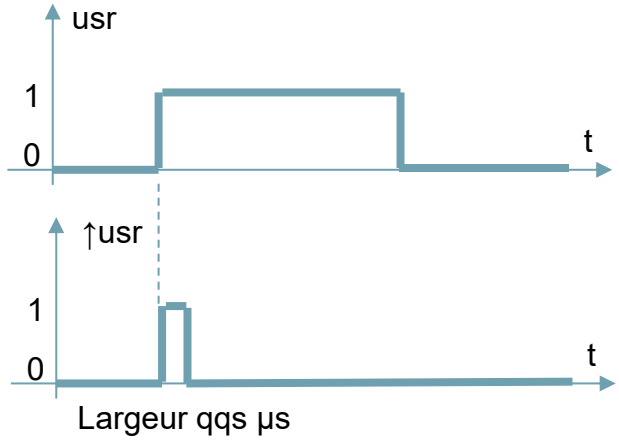
Gestion des évènements

Le terme usr est appelé **évènement**. Il peut être modélisé par le passage de 0 à 1 (en binaire) du signal usr (voir figure) appelé **front montant** noté ↑usr.

Le passage d'un état à un autre dans le graphe se réalise sur apparition d'évènements : si vous appuyez une fois sur usr, le passage de état1 à état2 se fait.

Le passage de l'état2 à l'état1 se fait 2s après l'activation de l'état2, autrement dit la led reste allumée 2s et attend un autre front de usr pour se rallumer.





Programme d'introduction 3/5

Détail du programme1

```
1  #programme 1 (validé)
2  ######## importation des bibliothèques
3  import machine,time #importation de la bibliothèque de la Pico (obligatoire)-
   et de time (gestion du temps)
4  #chargement sous bibliothèque Pin pour simplifier l'écriture du programme
5  from machine import Pin
6
7  ######## déclaration des entrées
8  #déclaration de la variable binaire associée au port 15 de la carte en mode
   entrée (voir carte E/S) pour le bouton usr
9  usr=Pin(15,Pin.IN,Pin.PULL_DOWN)
10
11 ######## déclaration sorties
12 #déclaration de la led verte interne de la carte en sortie
13 #Si pico
14 ledverte=Pin(25,Pin.OUT)
15 #Si pico W (wifi)
16 #ledverte=Pin("LED",Pin.OUT)
17
18 ######### initialisation constantes
19 etat=1  #c'est l'état 1 qui est actif au départ
20 etr=0   #etr : évènement temporel relatif
```

Le bouton usr est déclaré sur GP15. Le paramètre PULL_DOWN permet la mise à 0 rapide du signal en cas de relâchement du bouton.

En fonction du modèle de Pico la déclaration de la led verte interne est différente.

Programme d'introduction 4/5

Détail du programme1

```
22 ######## fonctions et interruptions (scrutations des
23 évènements)
24 #fonction appelée par une interruption du front montant de usr
25 #évolution graphe liée par des évènements
26 def detect_usr(ref):
27     global etat
28     global etr
29     if etat==1:
30         etat=2
31         etr=time.ticks_ms()
32     return
33
34 #création de l'interruption sur l'entrée usr
35 usr.irq(detect_usr,trigger=Pin.IRQ_RISING) #au front montant
   (rising) du signal reçu par usr la fonction detect_usr est
   lancée en priorité.
```

La procédure `detect_usr` doit être paramétrée par `ref`. Les variables `etat` et `etr` sont utilisées dans d'autres procédures et la boucle principale et doivent donc être déclarées en global.

`etr` mémorise la « date » courante c'est-à-dire le moment ou le front de usr est détecté. Il sera exploité car un évènement de type after sera à programmer.

Création d'une interruption associée à la procédure `detect_usr` précédemment créée, à l'entrée usr et au front montant du signal (RISING)*

L'interruption garantit la réaction de la carte dès l'occurrence de l'évènement associé au port. On peut créer autant d'interruptions que d'entrées distinctes.

* FALLING pour un front descendant

Programme d'introduction 5/5

Détail du programme1

```
37  ########### programme principal SED
38  while True:
39      #évolution graphe non liée par des évènements externes
40      if etat==2 and time.ticks_diff(time.ticks_ms(), etr) > 2000 :
    #mesure de l'écart temporel entre maintenant et etr en ms.
41          etat=1
42
43      #affectation des sorties
44      if etat==1:
45          ledverte.value(0) #Eteindre la led
46      if etat==2 :
47          ledverte.value(1) #Allumer la led
```

Boucle principale : ne s'interrompt que par un Ctrl-C dans le shell python.

On gère ici l'évènement temporel relatif apparaissant sous forme d'évènement after(2s) dans le graphe donc non traité par une interruption puisque en interne.

Affectation des sorties en fonction du numéro d'état courant.

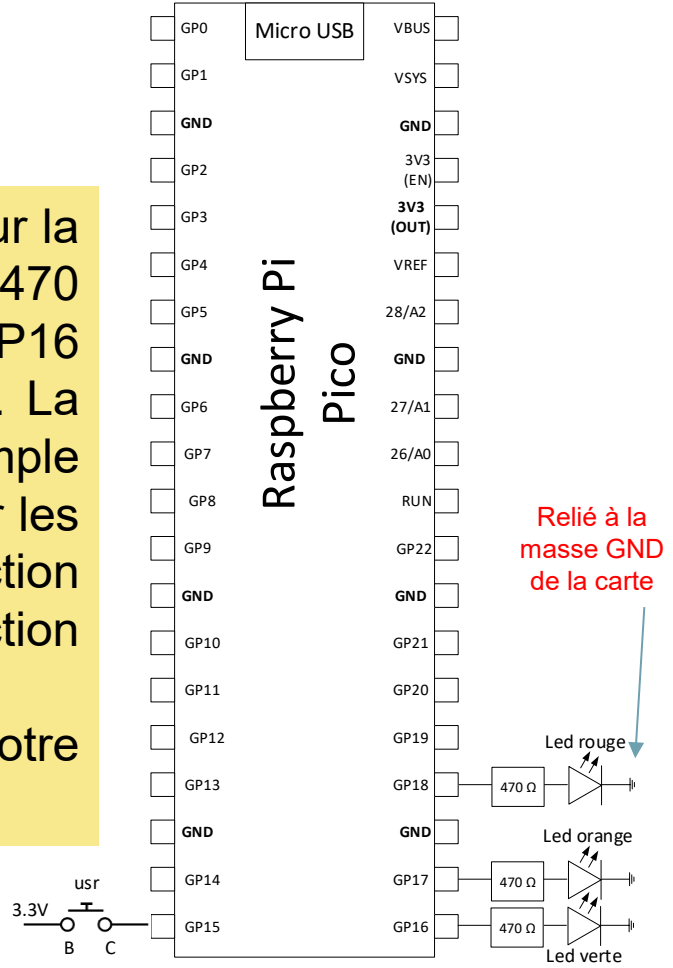
- Copiez le programme fourni (« sed_feux_1.py ») sur votre espace disque et l'ouvrir avec Thonny.

*Ce programme parait bien compliqué pour le peu qu'il fait : mais c'est un **programme générique**. Pour un diagramme d'états beaucoup plus complexe, le code évoluera à peine (si vous avez compris !).*

Branchement feux tricolores

- Vous devez brancher 3 leds (verte, orange, rouge) sur la carte de prototypage en plaçant des résistances de 470 Ω* entre la led et le port de sortie (par exemple GP16 pour la verte). La figure ci-jointe montre le câblage. La disposition des éléments est proposé à titre d'exemple évidemment. Sur vos plaques, je préconise de placer les boutons (ou entrées) sur la plaque de gauche (fonction acquérir) et les leds (ou sorties) à droite (fonction convertir et/ou moduler si hacheur).
- Vous pouvez tester et pensez à sauvegarder votre programme.

Pour rappel, le fil **court** de la led est l'anode et doit être branché sur la masse (GND).

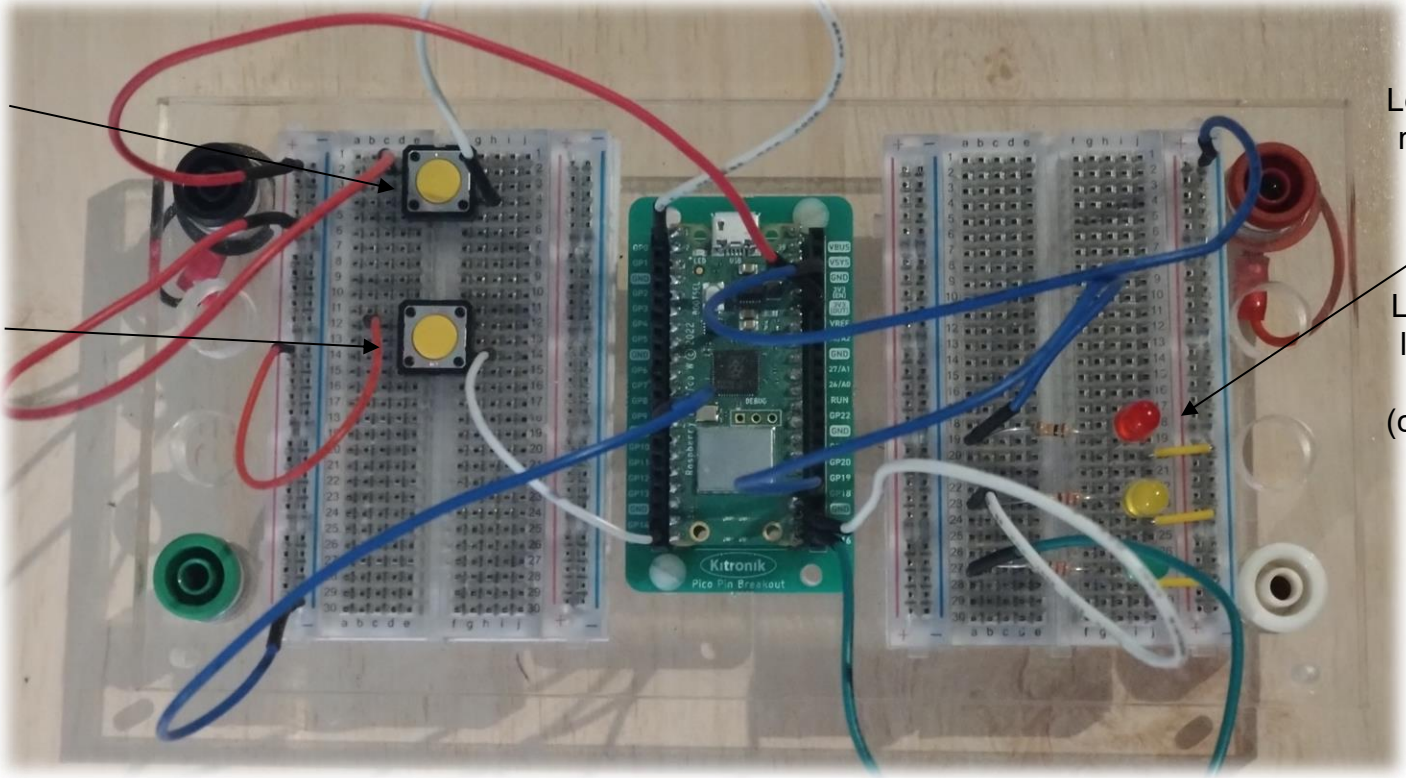


*Résistance 470 Ω : jaune violet marron + or

Branchement feux tricolores

Bouton trafic branché sur GP0 (voir plus loin)

Bouton usr branché sur GP15



Led rouge et sa résistance sur GP18

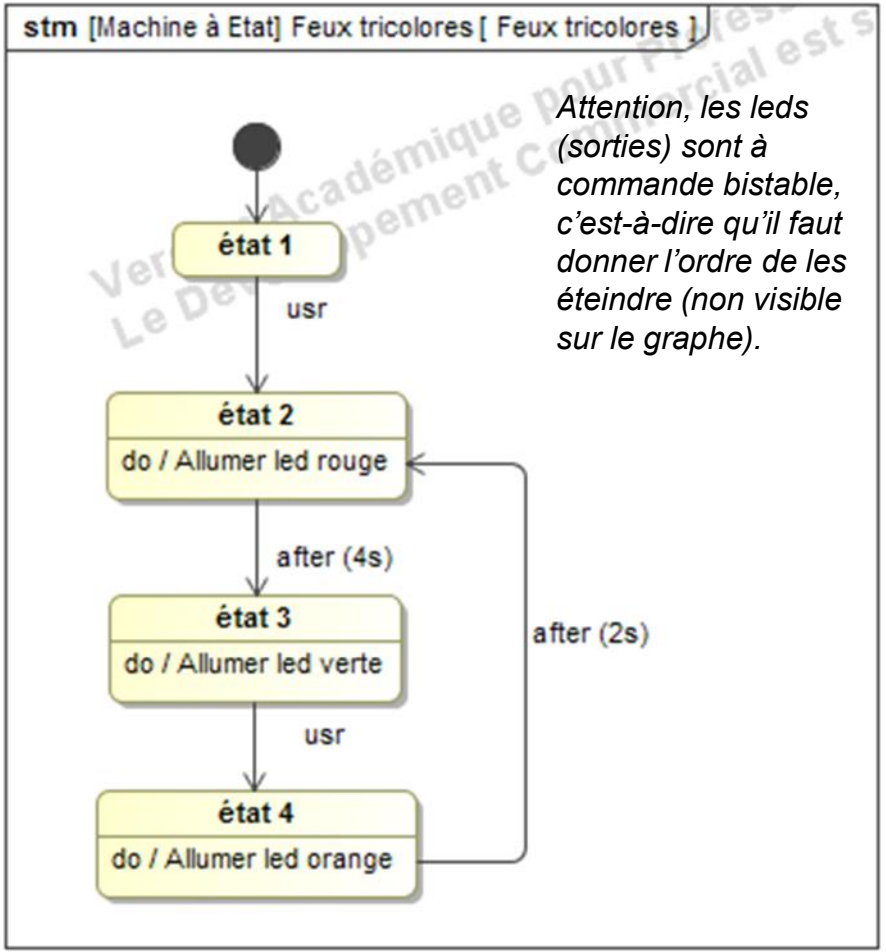
Le fil court de la led est reliée à la masse (colonne bleue à droite).

Programmes feux tricolores

- En complétant le programme d'introduction fourni précédemment, écrivez et testez le programme associé aux feux tricolores.

Pour réussir brillamment cette manipulation, vous devez analyser le lien entre le programme python fourni et le diagramme des questions précédentes. Pas de bricolage !

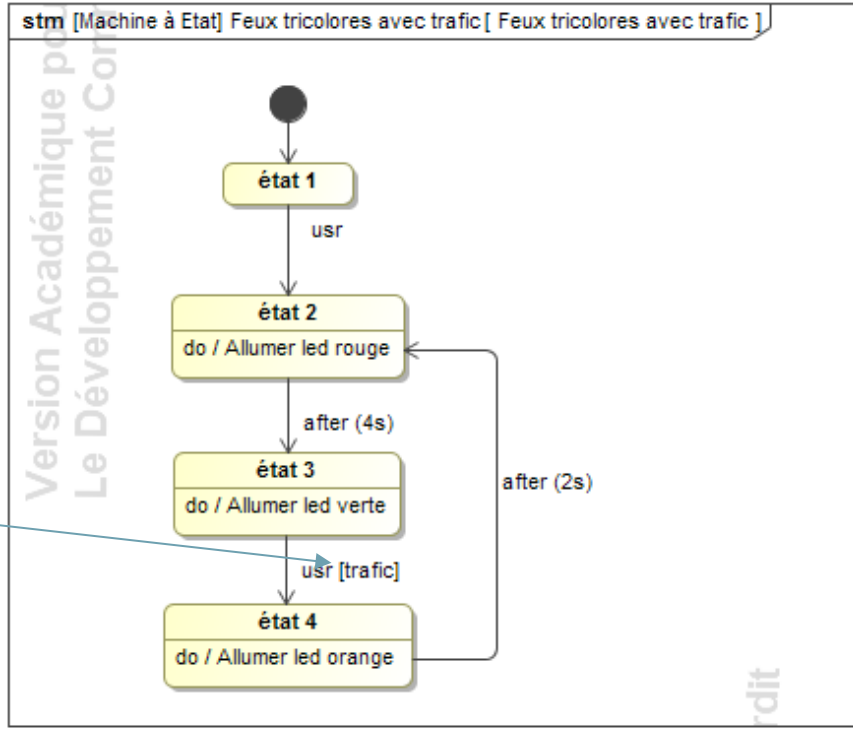
Rappels : dans le shell,
- Ctrl-C arrête le programme en cours et garde en mémoire le contenu des variables. Utile en phase de débogage.
- Ctrl-D effectue un soft reboot (équivalent au bouton STOP de Thonny)



Attention, les leds (sorties) sont à commande bistable, c'est-à-dire qu'il faut donner l'ordre de les éteindre (non visible sur le graphe).

Programmes feux tricolores et trafic

Modification du cahier des charges : la prise en compte de usr n'est possible que si le booléen « trafic » est à 1. Sur le diagramme cela apparait comme une **garde** (entre crochets).
L'évolution de l'état 3 à l'état 4 a lieu au moment où usr passe à 1 (front montant) à condition que « trafic » soit à 1.



- Câbler un nouveau bouton que l'on appellera « trafic » (comme usr sur l'entrée de votre choix)
- Modifier votre programme précédent afin de tenir compte de la garde [trafic].

Consulter l'aide si pb de branchement

Un nombre booléen est un nombre en base 2 : il n'a que deux valeurs possibles : 0 ou 1.

Programmes feux tricolores, trafic et mode de marche/arrêt

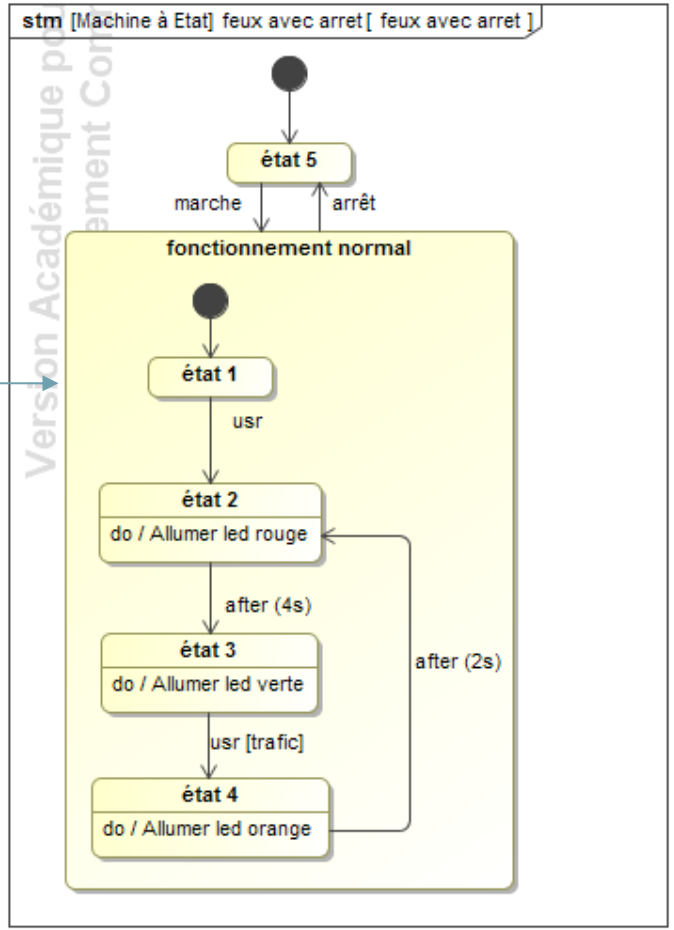
Modification du cahier des charges : on souhaite maintenant mettre en place deux modes de fonctionnement du système :

- Mode arrêt (état5)
- Mode fonctionnement normal (**état composite**)

Interprétation : A la mise en route, l'état5 est actif et les leds sont éteintes.

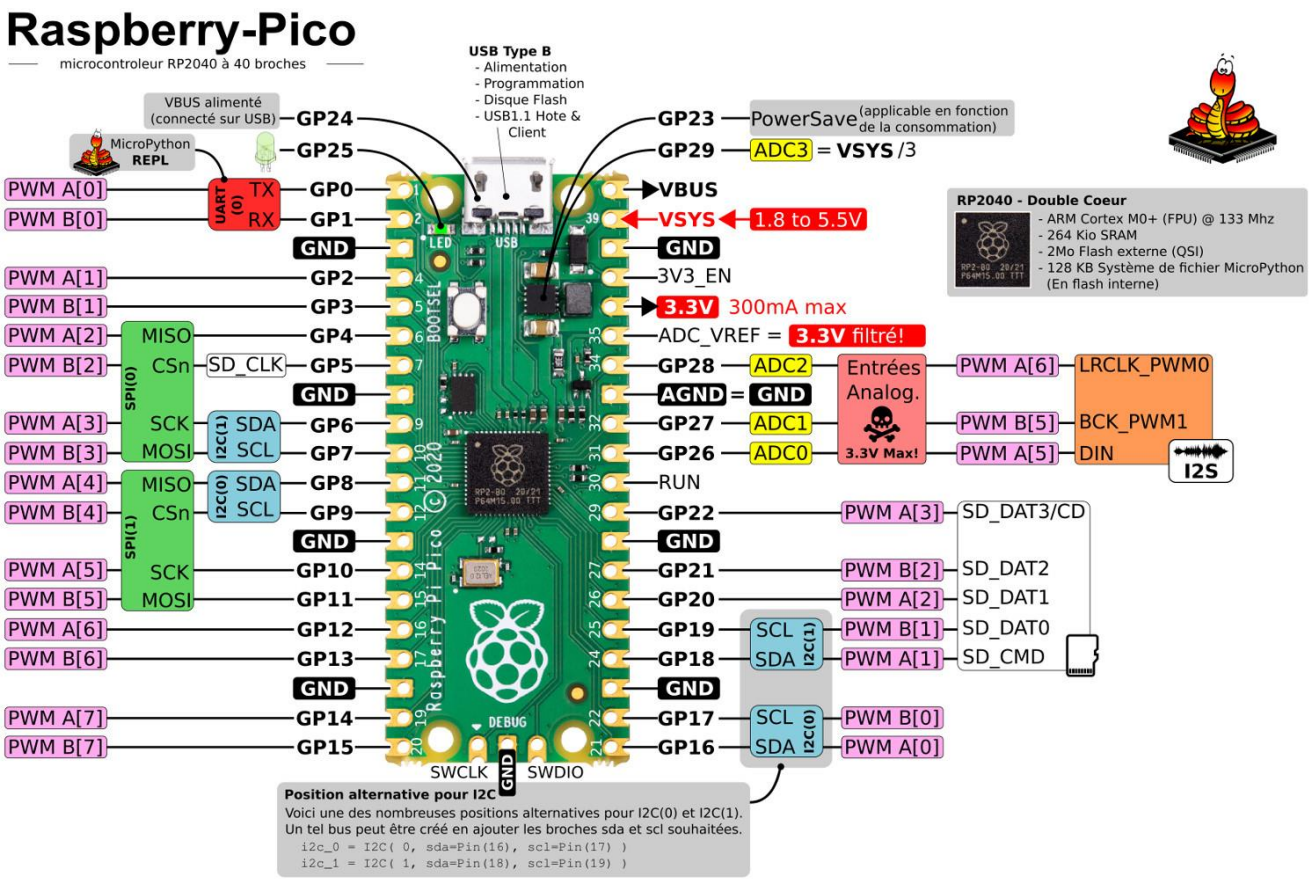
Lorsque l'évènement « marche » apparait, on active l'état1 (et désactive l'état5) et le diagramme suit son cours.

A tout moment, si l'évènement « arrêt » apparait, l'état5 s'active désactivant de fait n'importe quel état de « fonctionnement normal » et toutes les leds s'éteignent.



- Branchez deux boutons marche et arrêt supplémentaires et complétez votre programme précédent pour réaliser un système ayant un comportement modélisé par ce diagramme d'états.

Légendes des connecteurs de la Pico



Source : https://shop.mchobby.be/fr/