Feux de circulation - Simulation

Dans cet exercice, tu vas reproduire le fonctionnement **réaliste** de deux feux tricolores qui gèrent une intersection simple. L'objectif est de comprendre la **logique de sécurité** (priorités, phases, temps de dégagement) et de la traduire en **programme MicroPython** sur un **Pi Pico** en pilotant des LED.

4TTR Découverte

Fonctionnement d'un vrai carrefour à deux feux

Sur une route, deux directions qui se croisent ne peuvent pas être au vert en même temps, sinon collision.

Un carrefour à deux feux suit généralement des **phases** :

- **Phase A (direction A passe)** :
  - Feu A : **vert**
  - Feu B : **rouge**
- **Transition A → B** :
  - Feu A : **orange** (prépare l'arrêt)
  - Feu B : **rouge** (reste bloqué)
- **Temps de sécurité "tout rouge" (dégagement)** :
  - Feu A : **rouge**
  - Feu B : **rouge** Permet de laisser le temps à un véhicule engagé de terminer de traverser.
- **Phase B (direction B passe)** :
  - Feu B : **vert**
  - Feu A : **rouge**
- **Transition B → A** :
  - Feu B : **orange**
  - Feu A : **rouge**
- **Tout rouge** puis on recommence.

C'est exactement ce que tu vas programmer, avec des durées réalistes (ou accélérées pour la démo).

Objectif de la simulation sur Pi Pico

Tu dois piloter **2 feux tricolores** :

- **Feu A** : 1 LED rouge, 1 LED "orange", 1 LED verte
- **Feu B** : 1 LED rouge, 1 LED "orange", 1 LED verte

Le programme devra :

- enchaîner les phases correctement,
- garantir qu'il n'y a **jamais deux verts simultanés**,
- inclure un **temps "tout rouge"** entre les changements de priorité,
- tourner en boucle.

Matériel

- 1 Raspberry Pi Pico (ou Pico W)
- 2 feux de circulation
- PC avec Thonny + firmware MicroPython

Câblage à réaliser

Chaque feux doit être branché :

Tu choisis tes GPIO, mais tu dois fournir un tableau clair dans ton code, par exemple :

- Feu A : rouge = GP2, orange = GP3, vert = GP4
- Feu B : rouge = GP6, orange = GP7, vert = GP8

(ceci est un exemple : à toi de décider, mais ton code doit correspondre à ton montage)

Initialisation

Les broches, durées, phases,... doivent être initialisées dans des variables au début du programme: évite les "nombres magiques".

Cycle imposé

Le cycle à programmer (durées modifiables mais cohérentes) :

- A vert : 6 s
- A orange : 2 s
- Tout rouge : 1 s
- B vert : 6 s
- B orange : 2 s
- Tout rouge : 1 s → boucle

Règle critique : **pendant A vert, B doit être rouge**, et inversement.

Travail demandé

Programme minimal attendu

- Un fichier `main.py`
- Initialisation des 6 GPIO en sorties
- Une boucle infinie qui enchaîne les phases
- À chaque phase, tu dois :
  - allumer uniquement les LED correctes
  - éteindre toutes les autres
  - attendre la durée prévue (`sleep`)

Sécurité logique

Ton code doit être lisible et éviter les erreurs :

- prévoir une fonction ou une procédure simple du type “mettre le feu A en vert, feu B en rouge” (si tu n’as pas encore vu les fonctions, tu peux le faire avec des blocs répétés, mais proprement)
- ne jamais laisser une LED d’une phase précédente allumée par accident

Extensions (bonus)

Choisis une ou plusieurs améliorations :

- Afficher la phase en console (ex : `print("A_VERT / B_ROUGE")`)
- Mode “nuit” : clignotement orange sur les deux feux (en appuyant sur un bouton)
- Bouton “piéton” : quand on appuie, déclencher au prochain cycle un arrêt plus long au rouge dans une direction
- Ajouter un compte à rebours (console) pour chaque phase

Critères de réussite

- Les LED correspondent exactement aux phases demandées
- Les transitions sont correctes (vert → orange → rouge, et temps tout rouge)
- Aucun cas où les deux feux sont verts en même temps
- Code clair, commenté, et facile à modifier (durées et broches)