Modélisation et amélioration des systèmes de gestion du trafic routier

Safaa BOUMOUR **N°:MH020M**

Plan

Introduction

Partie 1 : Modélisation mathématique d'une voie de l'autoroute

- Variation du débit des véhicules en fonction de leurs vitesses

Partie 2 : Gestion de la priorité des véhicules dans les intersections urbaines

- Modélisation
  - Modèle de conducteur intelligent
  - Modèle d'un feu de circulation
- Simulation
- Optimisation
  - Feu de circulation à cycle dynamique
  - Synchronisation des véhicules autonomes

Conclusion et limitations

Introduction

Problématique

Comment améliorer la fluidité du trafic routier sur les autoroutes et dans les intersections urbaines afin de réduire les embouteillages ?

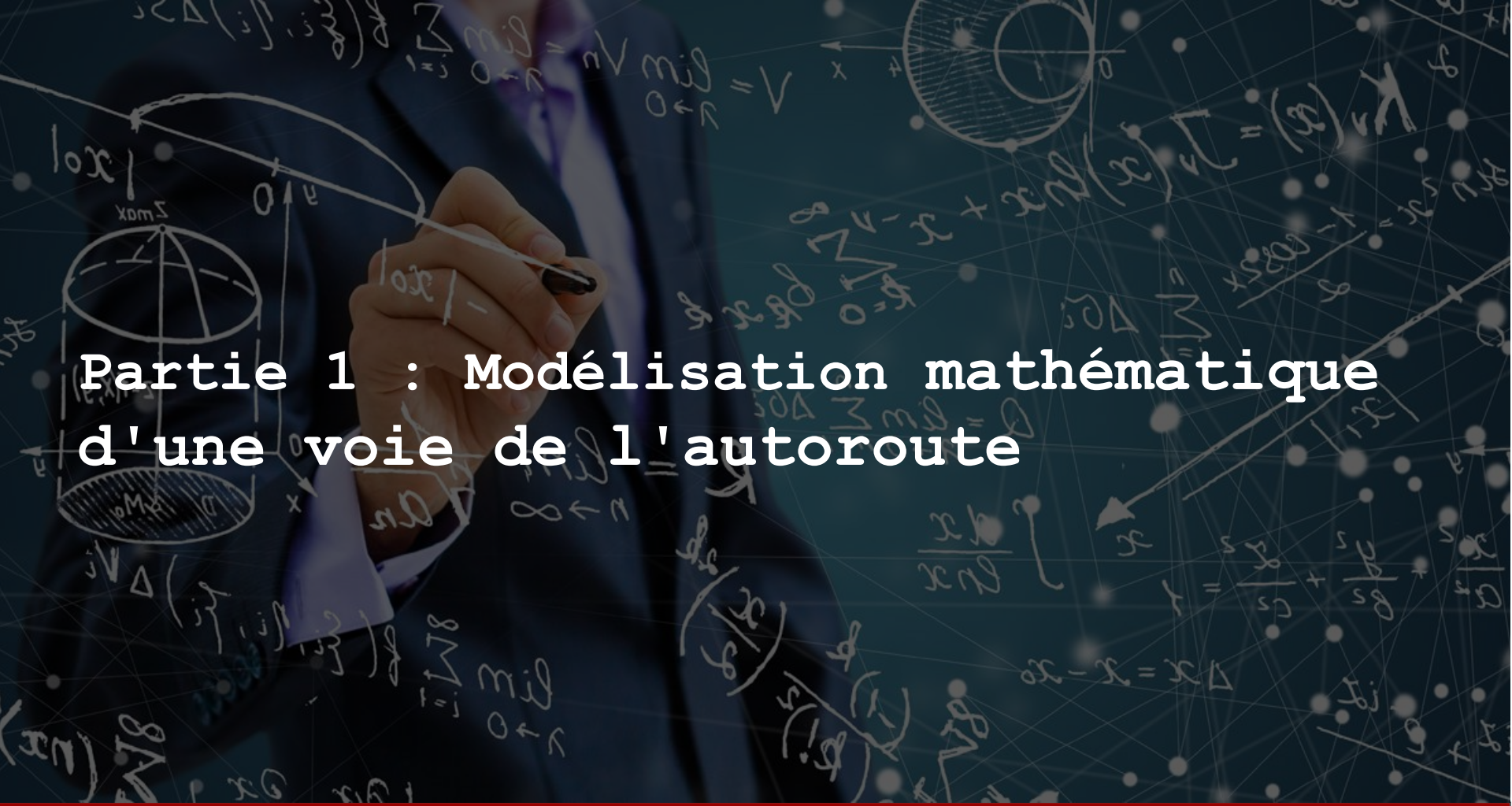
Objectifs

- **Modélisation mathématique du trafic routier en se basant sur les modèles candidats**

- Simulation rigoureuse et efficace de la gestion de la priorité des véhicules dans les intersections urbaines en Python.

- Créer des algorithmes d'amélioration des feux de circulation qui minimisent le temps mis par les véhicules pour traverser les carrefours.

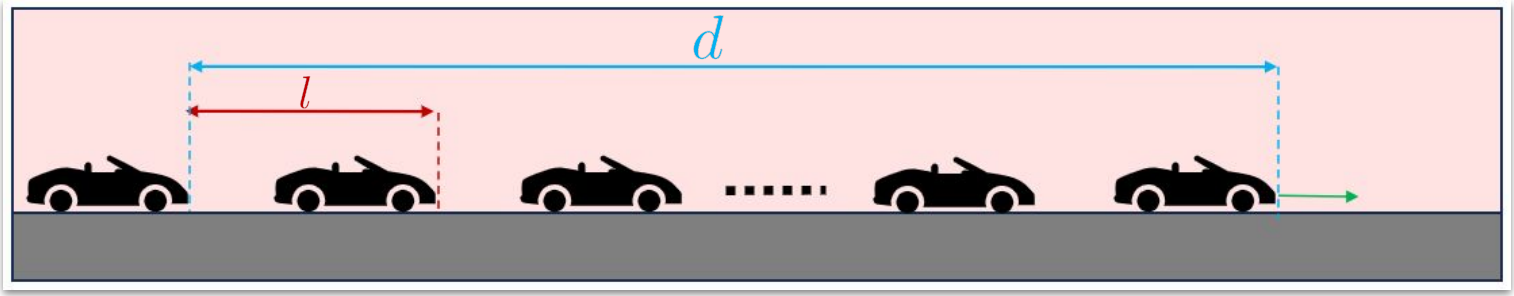
- Évaluation, vérification et comparaison des algorithmes.



Partie 1 : Modélisation mathématique d'une voie de l'autoroute

Modélisation

Modélisation mathématique d'une voie de l'autoroute



l : distance entre deux avant de véhicules

Modélisation

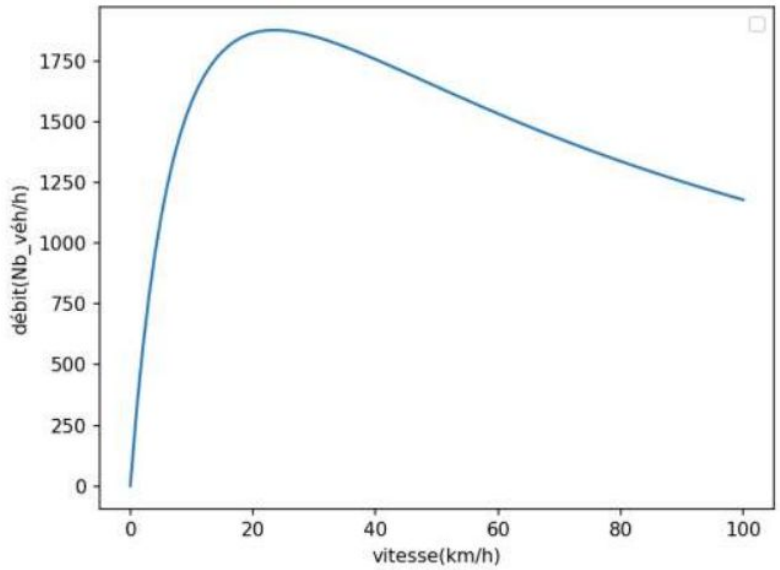
Modélisation mathématique d'une voie de l'autoroute

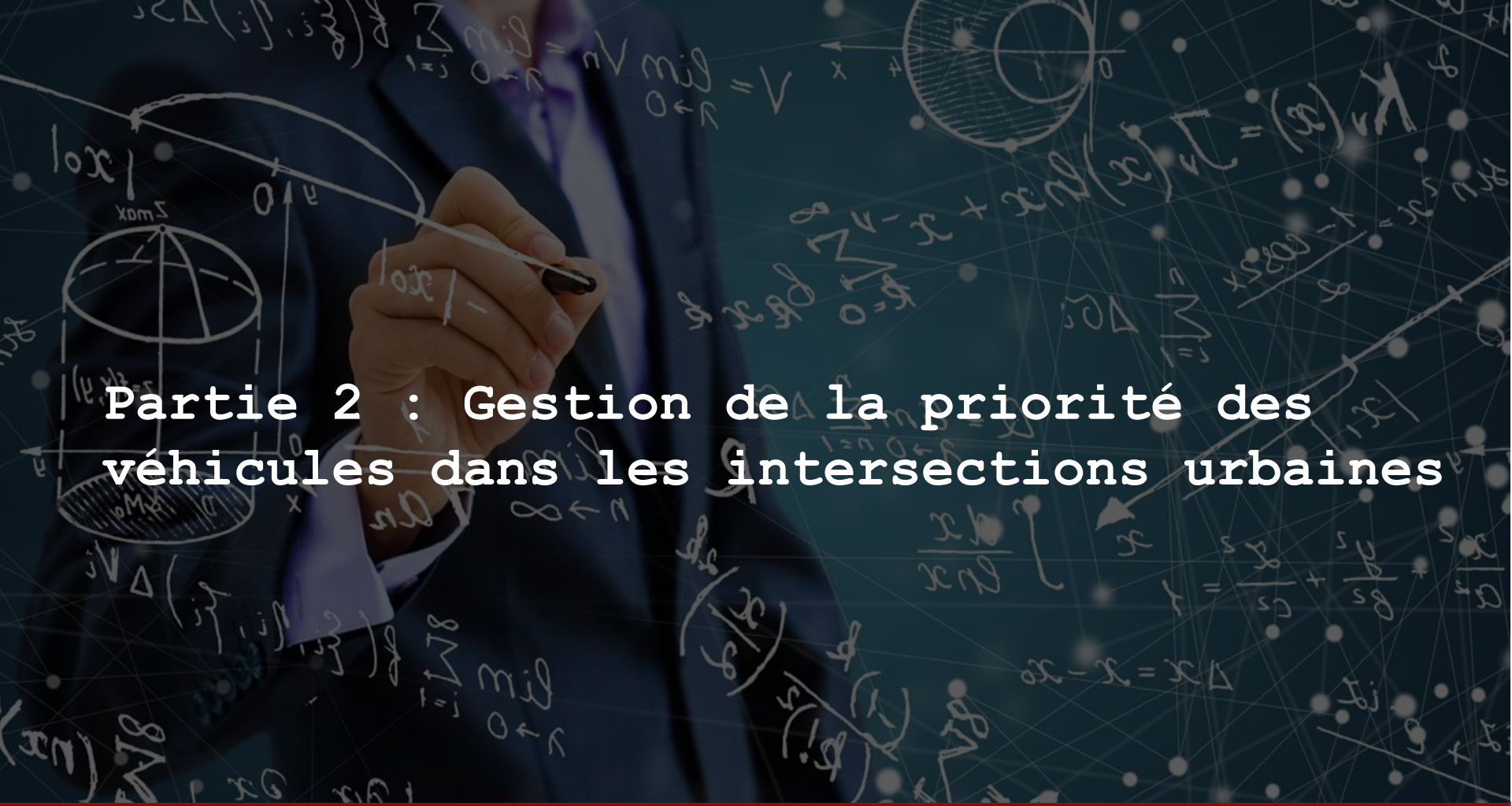
- Nombre de véhicules sur une portion de longueur d est égale : $N = \frac{d}{l}$
- $d = v.t$ avec $t = 1h = 3600s$, donc $d = 3600.v$
- Le débit (nombre de véhicules par heure) : $D(v) = \frac{d}{l} = \frac{3600.v}{l}$
- Distance de sécurité entre deux véhicules : $ds = c.v^2 + tr.v$
- La longueur du véhicule est L, alors : $l = c.v^2 + tr.v + L$

D'où : $$D(v) = \frac{3600.v}{c.v^2 + tr.v + L}$$

Débit en fonction de la vitesse

- On a : $$D(v) = \frac{3600.v}{c.v^2 + tr.v + L}$$
- Tracé de la courbe du débit $D(v)$, pour $c = 0.07s/m^2$, $tr = 1s$ et $L = 3m$.





Partie 2 : Gestion de la priorité des véhicules dans les intersections urbaines

Objectifs

- Modélisation mathématique du trafic routier en se basant sur les modèles candidats
- **Simulation rigoureuse et efficace de la gestion de la priorité des véhicules dans les intersections urbaines en Python.**
- Créer des algorithmes d'amélioration des feux de circulation qui minimisent le temps mis par les véhicules pour traverser les carrefours.
- Évaluation, vérification et comparaison des algorithmes

Modélisation

Modèle de conducteur intelligent

Hypothèses :

- Les véhicules ne changent pas de voie.
- Le $i^{éme}$ conducteur ne voit que le $(i-1)^{éme}$ conducteur.
- Pas de dépassement des véhicules en circulation

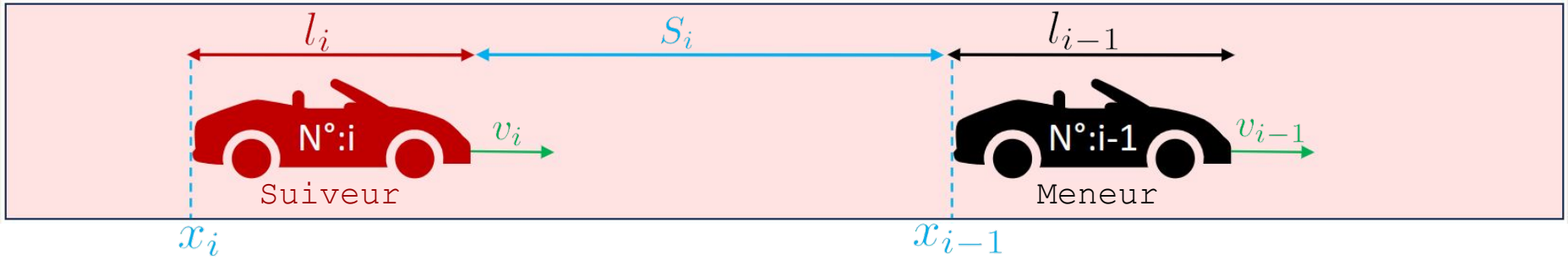
Définition :

Véhicule

$$\begin{cases} Identifiant & : & id \\ Longueur & : & l \\ Position\ actuelle & : & x_i \\ Vitesse\ actuelle & : & v_i \\ Vitesse\ souhaitée & : & v_{0,i} \\ Accélération\ maximale & : & a_i \\ Décélération\ confortable & : & b_i \\ Temps\ de\ réaction & : & T_i \\ Identifiant\ du\ meneur & : & id_{meneur} \\ Écart\ minimale & : & s_{0,i} \end{cases}$$

Modélisation

Modèle de conducteur intelligent



$$S_i = x_{i-1} - x_i - l_i$$

$$\Delta v_i = v_i - v_{i-1}$$

Modélisation

Modèle de conducteur intelligent

Mettre les véhicules en mouvement :

- $v_i(t + \Delta t) = v_i(t) + \frac{dv_i}{dt}\Delta t$
- $x_i(t + \Delta t) = x_i(t) + v_i(t)\Delta t + \frac{1}{2}\frac{dv_i}{dt}(\Delta t)^2$

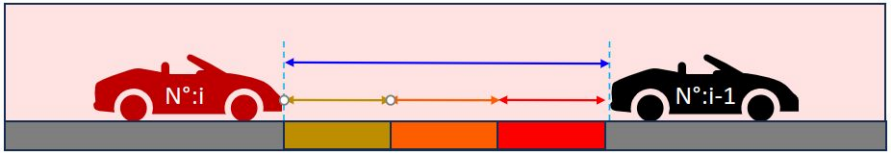
Modélisation

Modèle de conducteur intelligent

$$\frac{dv_i}{dt} = a_{route\_libre} + a_{interaction}$$

$$\begin{cases} a_{route\_libre} = a_i \left(1 - \left(\frac{v_i}{v_{0,i}}\right)^{\delta}\right) \\ a_{interaction} = -a_i \left(\frac{s^*(v_i, \Delta v_i)}{s_i}\right)^2 \end{cases}$$

avec δ : Exposant d'accélération

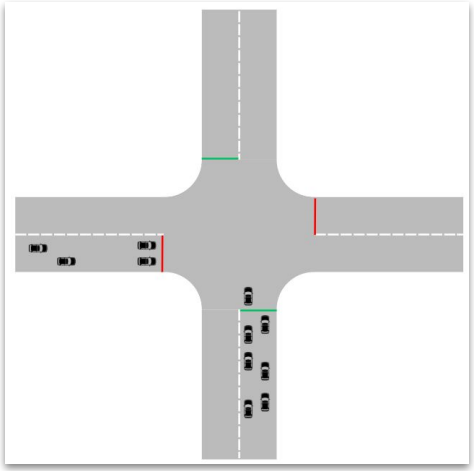


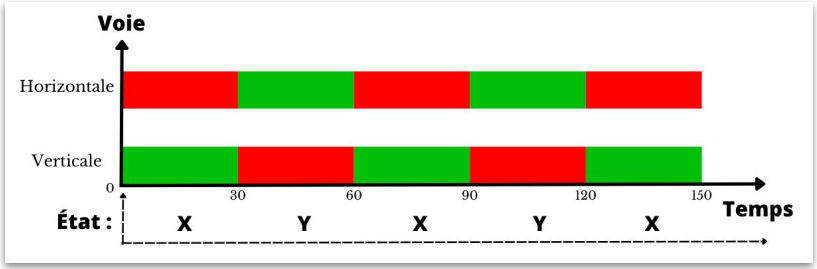
$$s^*(v_i, \Delta v_i) = \underbrace{v_i T_i}_{\text{Distance de réaction}} + \underbrace{\frac{v_i \Delta v_i}{\sqrt{2 a_i b_i}}}_{\text{Distance de freinage}} + \underbrace{s_{0,i}}_{\text{Écart minimal}}$$

Distance de sécurité

Modélisation

Feu de circulation à cycle constant symétrique





Fonction d'état d'un feu de circulation

Hypothèses

Les feux de circulation :

- Ne comportent que deux voies : horizontales et verticales
- Oscillent entre les états **X** et **Y** successivement et indéfiniment.

Définitions

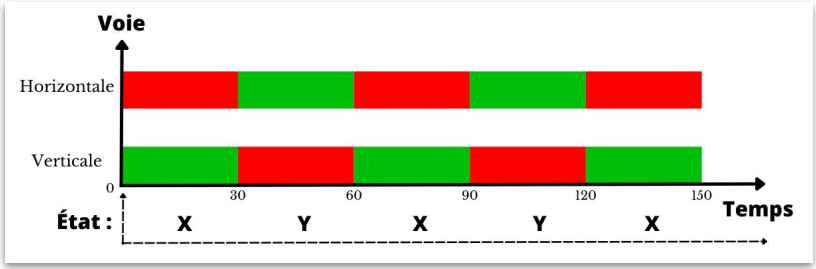
Durées de chaque état du feu de circulation :

- $T_X(t)$: Durée de l'état **X** (Feu vert dans la voie verticale).
- $T_Y(t)$: Durée de l'état **Y** (Feu vert dans la voie horizontale).

$$T_X(t) = T_Y(t) = T$$

Modélisation

Feu de circulation à cycle constant symétrique



Fonction d'état d'un feu de circulation à cycle constant symétrique

Algorithm 1 Switch des feux de circulation pour les chemins X et Y

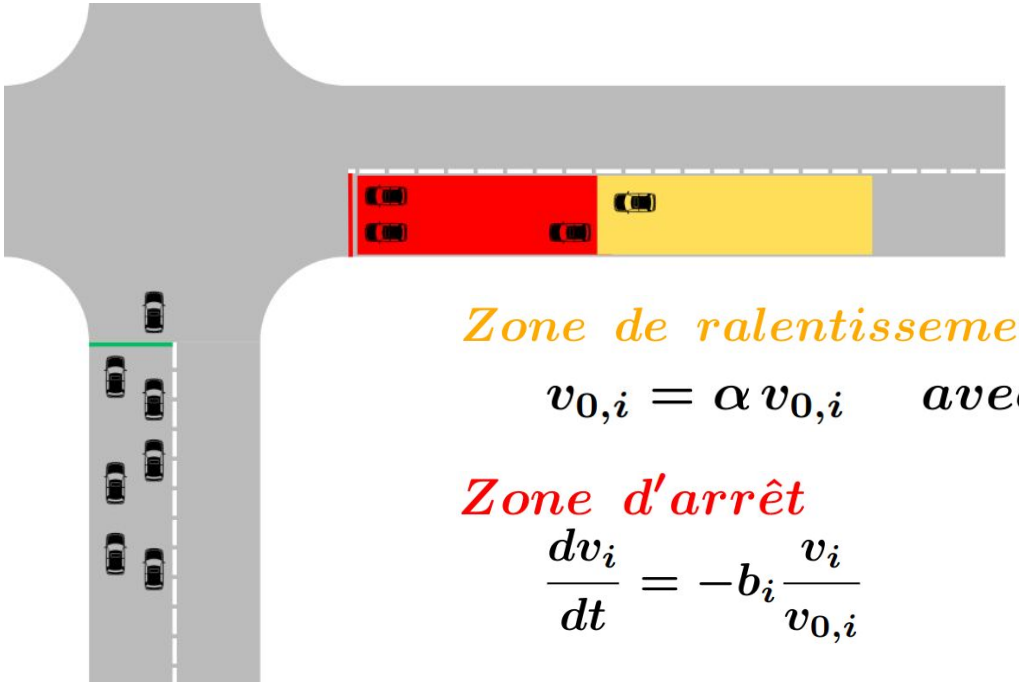
```
1: global variables
2:     tempsDernierChangement = 0
3:     dernierCheminVert = 'X'
4: end global variables
5: function SWITCH_FEU(t)
6:     tempsDernierChangement = t
7:     if tempsDernierChangement == 'X' then
8:         return 'Y'
9:     end if
10:    return 'X'
11: end function
```

Algorithm 2 Switch des feux de circulation à cycle constant symétrique

```
1: global variables
2:     tempsDernierChangement = 0
3:     dernierCheminVert = 'X'
4:     periode = 30
5: end global variables
6: function CHEMIN_VERT_CONSTANTE(t)
7:     if t - tempsDernierChangement ≤ periode then
8:         return dernierCheminVert
9:     end if
10:    return SWITCH_FEU(t)
11: end function
```

Modélisation

Influence du feu de circulation sur la vitesse des véhicules



Zone de ralentissement

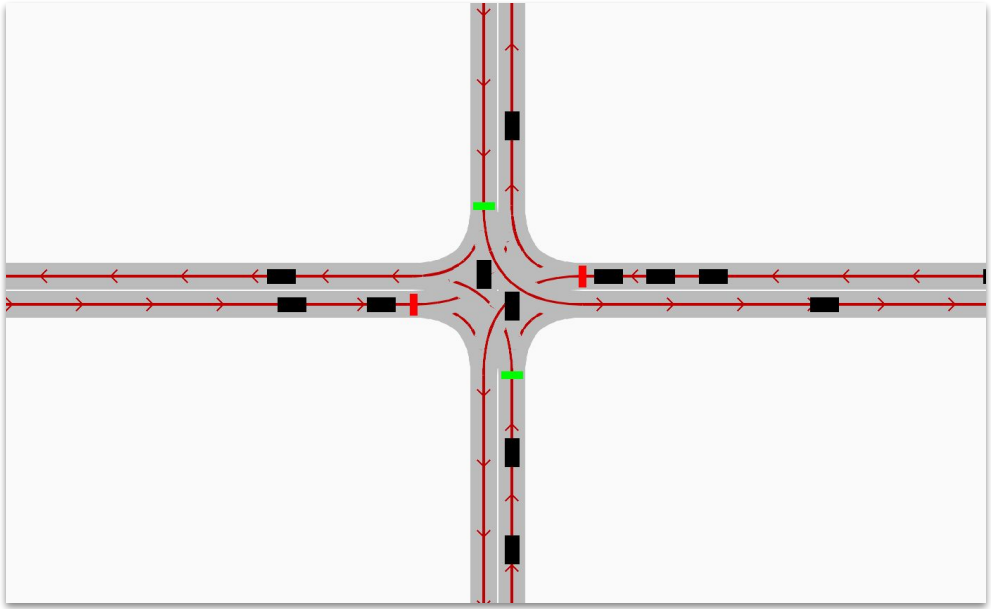
$$v_{0,i} = \alpha\, v_{0,i} \qquad avec\ \alpha < 1$$

Zone d'arrêt

$$\frac{dv_i}{dt} = -b_i \frac{v_i}{v_{0,i}}$$

Simulation

Simulation en python à l'aide de la bibliothèque pygame



Simulation en python à l'aide de la bibliothèque pygame

Paramètres de la simulation :

- Les distances minimales entre les véhicules sont les mêmes : $s_{0,i} = 1.5m$
- Les vitesses souhaitées sont les mêmes :

$$v_{0,i} = 50km/h = 13.9m/s$$

- Durée de feu de circulation :

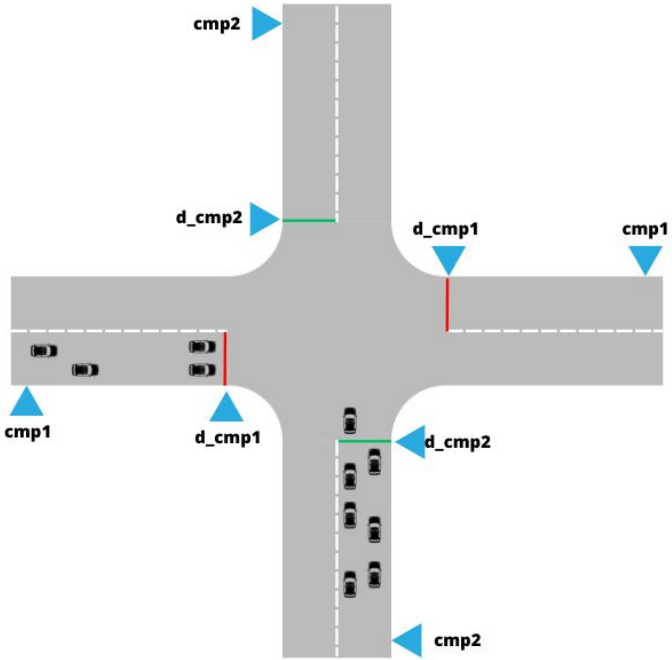
$$T_X(t) = T_Y(t) = 30s$$

Objectifs

- Modélisation mathématique du trafic routier en se basant sur les modèles candidats
- Simulation rigoureuse et efficace de la gestion de la priorité des véhicules dans les intersections urbaines en Python.
- **Créer des algorithmes d'amélioration des feux de circulation qui minimisent le temps mis par les véhicules pour traverser les carrefours.**
- Évaluation, vérification et comparaison des algorithmes

Optimisation

Solution 1 : feu de circulation à cycle dynamique



Hypothèses

Les capteurs :

- Chaque feu de circulation est accompagné par deux capteurs, pour compter le nombre des véhicules qui attendent le feu de circulation

Définitions

Durées de chaque état du feu de circulation :

- $T_X(t)$: Durée de l'état **X** (Feu vert dans la voie verticale).
- $T_Y(t)$: Durée de l'état **Y** (Feu vert dans la voie horizontale).

$$T_X(t) \neq T_Y(t)$$

$$\begin{cases} 20s \leq T_X(t) \leq 40s \\ 20s \leq T_Y(t) \leq 40s \end{cases}$$

Nombre des véhicules qui attendent le feu de circulation :

- $N_X(t)$: Pour la voie verticale.
- $N_Y(t)$: Pour la voie horizontale.

Optimisation

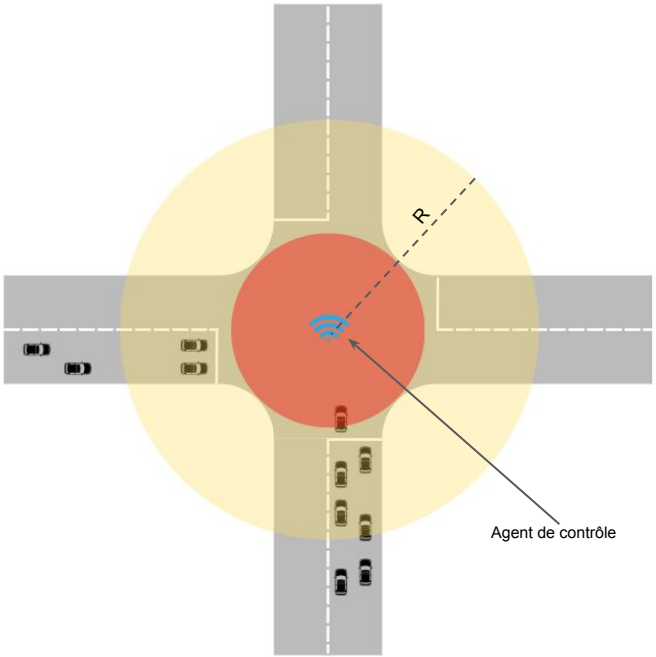
Solution 1 : simulation du feu de circulation à cycle dynamique

Algorithm 3 Switch des feux de circulation à cycle dynamique

```
1:  global variables
2:      tempsDernierChangement = 0
3:      dernierCheminVert = 'X'
4:      dureeMin = 20                    ▷ t min avant l'optimisation dynamique
5:      dureeMax = 40                    ▷ t max avant de swith les feux
6:  end global variables
7:  function CHEMIN_VERT_DYNAMIQUE(t, nbrX, nbrY)
8:      if (t - tempsDernierChangement > dureeMax) or
          (t - tempsDernierChangement > dureeMin and
          (nbrX > nbrY and dernierCheminVert == 'Y' or
          nbrX < nbrY and dernierCheminVert == 'X'))
           then
9:          return SWITCH_FEU(t)
10:     end if
11:     return dernierCheminVert
12: end function
```

Optimisation

Solution 2 : Synchronisation des véhicules autonomes



Hypothèses

- Un agent de contrôle est installé dans le centre de toutes les intersections.
- Tous les véhicules sont autonomes

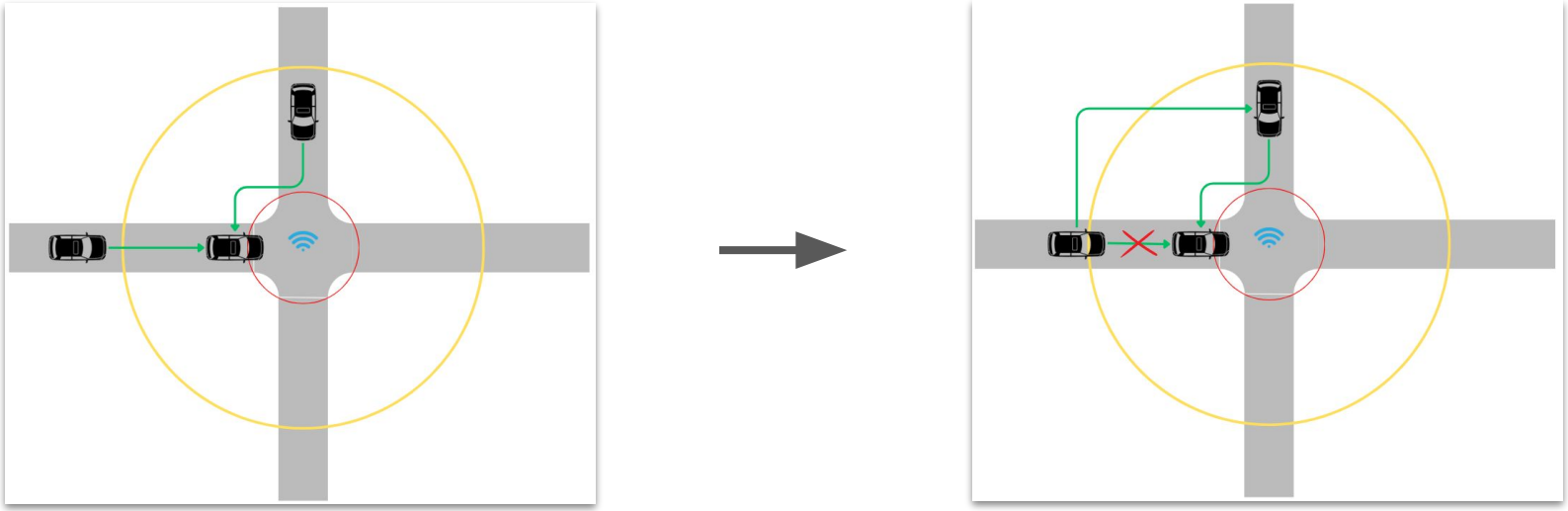
Définitions

Chaque intersection est caractérisée par :

- Une zone de synchronisation de rayon R

Optimisation

Solution 2 : Principe de synchronisation des véhicules

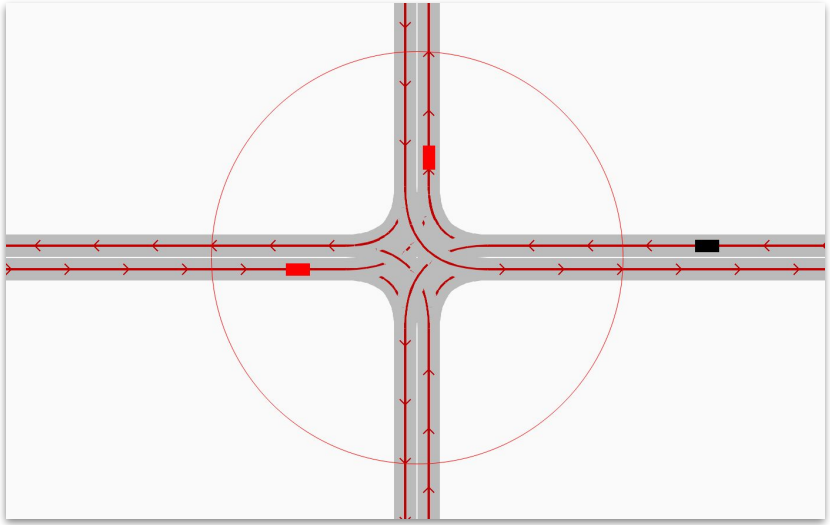


Avant qu'un véhicule entre dans la zone de synchronisation

Après qu'un véhicule entre dans la zone de synchronisation

Optimisation

Solution 2 : Simulation de la synchronisation des véhicules



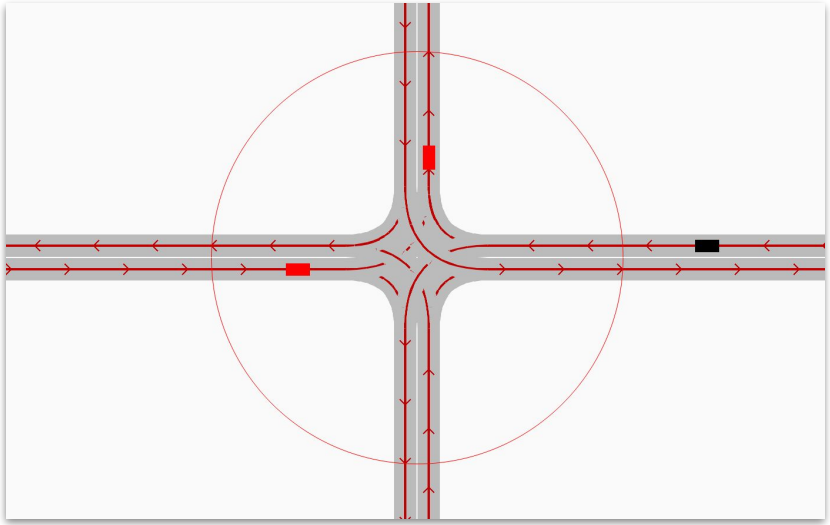
Simulation de la synchronisation en python à l'aide de la bibliothèque pygame

Paramètres de simulation

- La simulation est sur deux voies bidirectionnelles
- Chaque voie est de longueur 100m
- le rayon R de l'agent de contrôle est fixé à 30m
- Tous les véhicules ayant même longueur : 4m
- Le pas de simulation est 1/60s

Optimisation

Solution 2 : Simulation de la synchronisation des véhicules



Simulation de la synchronisation en python à l'aide de la bibliothèque pygame

Algorithm 4 Changer meneur - Synchronisation des véhicules autonomes

```
1: function CHANGER_MENEUR(vehicules, vehicule)
2:     if len(vehicules) > 0 then
3:         vehicule.meneur = vehicules[0].id
4:     end if
5: end function
```

Algorithm 5 Détection véhicule - Synchronisation des véhicules autonomes

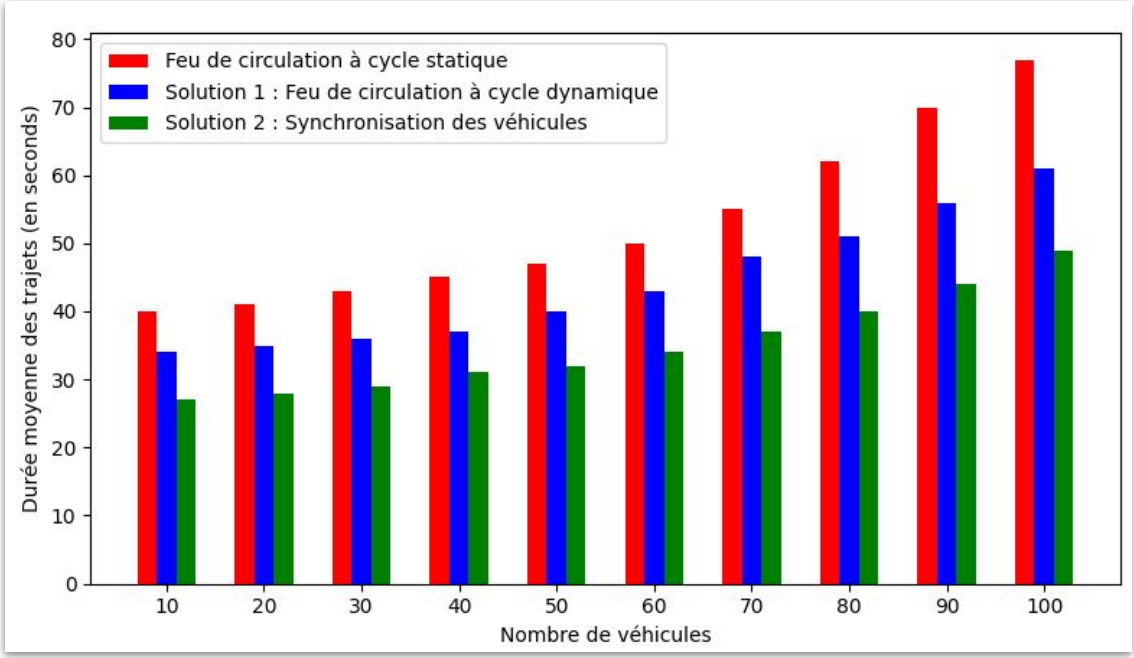
```
1: function EST_DANS_LA_ZONE(vehicules, vehicule, rayon, longueurRoute)
2:     if vehicule.x > (longueurRoute - rayon) and vehicule ∉ vehicules then
3:         COLORER_VEHICULE(vehicule)
4:         CHANGER_MENEUR(vehicules, vehicule)
5:         AJOUTER_VEHICULE(vehicules, vehicule)
6:     end if
7: end function
```

Objectifs

- Modélisation mathématique du trafic routier en se basant sur les modèles candidats
- Simulation rigoureuse et efficace de la gestion de la priorité des véhicules dans les intersections urbaines en Python.
- Créer des algorithmes d'amélioration des feux de circulation qui minimisent le temps mis par les véhicules pour traverser les carrefours.
- **Évaluation, vérification et comparaison des algorithmes.**

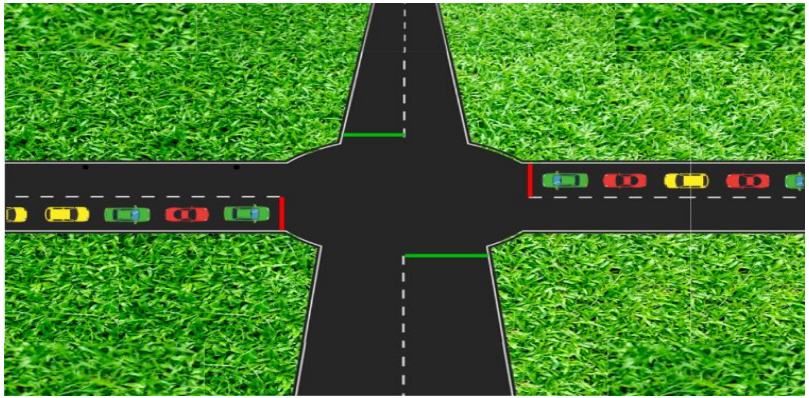
Comparaison

Comparaison et évaluation des différentes solutions établies



Mise en Comparaison des différentes solutions à l'aide d'un histogramme en python avec Matplotlib.

Conclusion et limitations



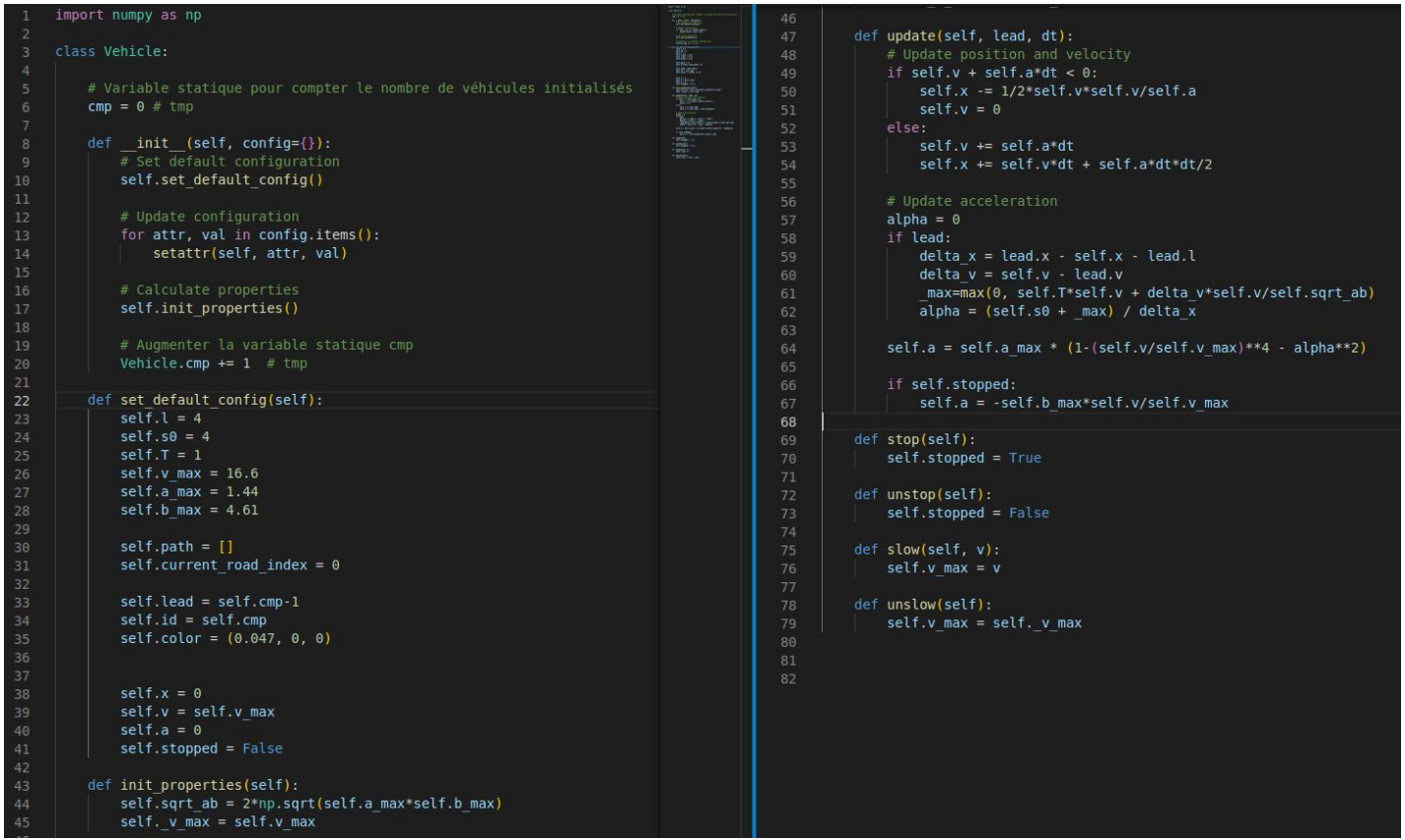
- 🌟 Feux de circulation à cycle statique :❌
- 🌟 Feux de circulation à cycle dynamique :❌
- 🌟 Synchronisation des véhicules :✅

- 🌟 Feux de circulation à cycle statique :❌
- 🌟 Feux de circulation à cycle dynamique : ✅
- 🌟 Synchronisation des véhicules : ✅

Annexe 1 : code Python de l'histogramme

```python
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(10, 110, 10)  # Nombre de véhicules dans chaque simulation : 10, 20, ..., 100

# Durées moyenne des trajets (en seconds) calculés à partir de la simulation des trois Modèles
y1 = [40, 41, 43, 45, 47, 50, 55, 62, 70, 77]  # Modèle 1 - Feux rouges statiques
y2 = [34, 35, 36, 37, 40, 43, 48, 51, 56, 61]  # Modèle 2 - Feux rouges dynamiques
y3 = [27, 28, 29, 31, 32, 34, 37, 40, 44, 49]  # Modèle 3 - Synchronisation des véhicules

# Configuration des couleurs
couleurs = ['red', 'blue', 'green']  # Couleurs pour les barres correspondant à y1, y2 et y3

# Création des positions des barres sur l'axe des x
positions = np.arange(len(x))

# Largeur des barres
largeur = 0.2  # Vous pouvez ajuster cette valeur pour modifier l'espacement entre les barres

# Tracer l'histogramme
plt.bar(positions, y1, width=largeur, color=couleurs[0], label='Feu de circulation à cycle statique')
plt.bar(positions + largeur, y2, width=largeur, color=couleurs[1], label='Solution 1 : Feu de circulation à cycle dynamique')
plt.bar(positions + 2*largeur, y3, width=largeur, color=couleurs[2], label='Solution 2 : Synchronisation des véhicules')

# Configurer les étiquettes des axes
plt.xticks(positions + largeur, x)
plt.xlabel('Nombre de véhicules')
plt.ylabel('Durée moyenne des trajets (en seconds)')

# Ajouter une légende
plt.legend()

# Afficher le graphique
plt.show()
```

Annexe 2 : code Python de la classe véhicule



```python
import numpy as np

class Vehicle:

    # Variable statique pour compter le nombre de véhicules initialisés
    cmp = 0 # tmp

    def __init__(self, config={}):
        # Set default configuration
        self.set_default_config()

        # Update configuration
        for attr, val in config.items():
            setattr(self, attr, val)

        # Calculate properties
        self.init_properties()

        # Augmenter la variable statique cmp
        Vehicle.cmp += 1  # tmp

    def set_default_config(self):
        self.l = 4
        self.s0 = 4
        self.T = 1
        self.v_max = 16.6
        self.a_max = 1.44
        self.b_max = 4.61

        self.path = []
        self.current_road_index = 0

        self.lead = self.cmp-1
        self.id = self.cmp
        self.color = (0.047, 0, 0)


        self.x = 0
        self.v = self.v_max
        self.a = 0
        self.stopped = False

    def init_properties(self):
        self.sqrt_ab = 2*np.sqrt(self.a_max*self.b_max)
        self._v_max = self.v_max

    def update(self, lead, dt):
        # Update position and velocity
        if self.v + self.a*dt < 0:
            self.x -= 1/2*self.v*self.v/self.a
            self.v = 0
        else:
            self.v += self.a*dt
            self.x += self.v*dt + self.a*dt*dt/2

        # Update acceleration
        alpha = 0
        if lead:
            delta_x = lead.x - self.x - lead.l
            delta_v = self.v - lead.v
            _max=max(0, self.T*self.v + delta_v*self.v/self.sqrt_ab)
            alpha = (self.s0 + _max) / delta_x

        self.a = self.a_max * (1-(self.v/self.v_max)**4 - alpha**2)

        if self.stopped:
            self.a = -self.b_max*self.v/self.v_max

    def stop(self):
        self.stopped = True

    def unstop(self):
        self.stopped = False

    def slow(self, v):
        self.v_max = v

    def unslow(self):
        self.v_max = self._v_max
```