Université Sidi Mohamed Ben Abdellah de Fès La Faculté des Sciences Dhar El Mahraz

Compte rendu sur construire et lancer une première simulation SUMO pour les Smart Cities

Réalisé par **Bakir Mohammed**

Master **: Mlaim**

Enseignant: Pr.Noureddine EN-NAHNAHI

Introduction:

SUMO (Simulation of Urban Mobility) est un outil open-source de microsimulation utilisé pour modéliser et analyser les systèmes de transport urbain. Ce tutoriel présente une méthodologie complète pour construire et lancer une première simulation SUMO, mettant en avant son utilité dans le cadre des Smart Cities. Les objectifs principaux incluent la modélisation du trafic routier, l'analyse des émissions polluantes, et l'optimisation des infrastructures via des capteurs virtuels et des algorithmes adaptatifs.

Préparation du Réseau :

Les fichiers region.nod.xml et region.edg.xml définissent les intersections et les tronçons routiers.

```
<nodes>
    <node id="n1" x="0.0" y="0.0" type="priority"/>
    <node id="n2" x="100.0" y="0.0" type="traffic_light"/>
    <node id="n3" x="200.0" y="0.0" type="priority"/>
</nodes>
```

```
<edges>
    <edge id="edge1" from="n1" to="n2" numLanes="1" speed="13.9"/>
    <edge id="edge2" from="n2" to="n3" numLanes="1" speed="13.9"/>
</edges>
```

Après cela génère un réseau binaire (region.net.xml) incluant la géométrie des voies et la logique des feux tricolores en utilisant la commande **netconvert --node-files region.nod.xml --edge-files region.edg.xml -o region.net.xml.**

Génération du Trafic :

Le fichier region.rou.xml permet de définir des flux de véhicules (ex : 200 voitures injectées en 360 secondes).

```
<routes>
   <!-- Définition du type de véhicule -->
   <vType id="car" accel="2.0" decel="4.5" sigma="0.5" length="5.0"
        minGap="2.5" maxSpeed="13.9"
emissionClass="HBEFA3/PC_G_EU4"/>

   <!-- Génération d'un flow de 200 voitures -->
   <flow id="car_flow" type="car" begin="0" end="360" number="200"
        from="edge1" to="edge2"/>
</routes>
```

Instrumenter le scénario :

Le fichier **`region.add.xml`** installe deux capteurs :

```
<additional>
   <!-- Capteur de file d'attente sur edge1 -->
   <laneAreaDetector id="queue_detector_edge1" lane="edge1_0" pos="0"
length="100"
                freq="1"
                file="queue_detector_output.xml"
                friendlyPos="true"
                outputFormat="XML"/>

   <!-- Capteur de pollution sur edge1 entre 30m et 70m -->
   <emissionDetector id="pollution_detector" lane="edge1_0"
                pos="30"
                length="40"
                freq="1"
                file="pollution_output.xml"/>
</additional>
```

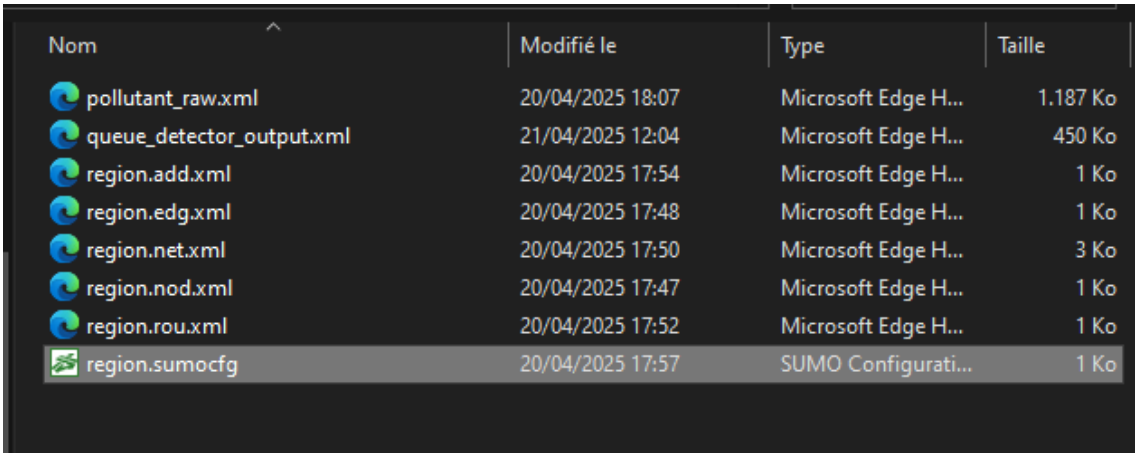
Créer le fichier de configuration :

Un fichier `region.sumocfg` regroupe tous les ingrédients :

```
<configuration>
  <input>
    <net-file value="region.net.xml"/>
    <route-files value="region.rou.xml"/>
    <additional-files value="region.add.xml"/>
  </input>
  <time>
    <begin value="0"/>
    <end   value="720"/>
    <step-length value="1"/>
  </time>
</configuration>
```

Lancer la simulation :

Apres le lancement de la simulation en utilisant la commande `sumo-gui -c region.sumocfg :`



Exploiter les résultats :

Chargement les données :

```
# Charger le fichier XML
tree = ET.parse("D:\\smartCities\\Trafic_Routier_Polution\\pollutant_raw.xml")
root_p = tree.getroot()
# Charger le fichier
tree = ET.parse("D:\\smartCities\\Trafic_Routier_Polution\\queue_detector_output.xml")
root_q = tree.getroot()
```

```python
data = []

# Extraire les données
for timestep in root_p.findall("timestep"):
    time = float(timestep.attrib["time"])
    for vehicle in timestep.findall("vehicle"):
        data.append({
            "time": time,
            "id": vehicle.attrib["id"],
            "speed": float(vehicle.attrib.get("speed", 0)),
            "CO": float(vehicle.attrib.get("CO", 0)),
            "CO2": float(vehicle.attrib.get("CO2", 0)),
            "NOx": float(vehicle.attrib.get("NOx", 0)),
            "PMx": float(vehicle.attrib.get("PMx", 0)),
            "fuel": float(vehicle.attrib.get("fuel", 0)),
            "noise": float(vehicle.attrib.get("noise", 0))
        })
```

```python
data_q = []

# Extraire les données
for interval in root_q.findall("interval"):
    data_q.append({

        "time_start": float(interval.attrib["begin"]),
        "time": float(interval.attrib["end"]),
        "meanSpeed": float(interval.attrib.get("meanSpeed", 0)),
        "haltingVehicles": int(interval.attrib.get("haltingVehicles", 0)),
        "waitingTime": float(interval.attrib.get("waitingTime", 0)),
        "occupancy": float(interval.attrib.get("occupancy", 0))
    })
```

Le code lit deux fichiers XML :

- Un contenant des **données de pollution des véhicules**
- L'autre contenant des **données sur le trafic routier**

Il extrait ensuite :

- Pour chaque véhicule : sa vitesse, ses émissions (CO, CO_2, etc.), bruit et carburant consommé
- Pour chaque intervalle de temps : la vitesse moyenne, les véhicules arrêtés, le temps d'attente et l'occupation de la route

Enfin, il stocke toutes ces données dans deux listes (data et data_q).

```python
# Convertir en DataFrame
df = pd.DataFrame(data)


# Afficher les premières lignes
df.head()
```

	time	id	speed	CO	CO2	NOx	PMx	fuel	noise
0	0.0	car_flow.0	0.00	164.78	2624.72	1.20	0.07	837.22	55.94
1	1.0	car_flow.0	2.00	147.86	3436.64	1.53	0.08	1096.17	65.49
2	2.0	car_flow.0	3.69	131.06	3875.50	1.69	0.08	1236.14	64.84
3	3.0	car_flow.0	4.98	112.41	3853.43	1.65	0.07	1229.09	63.71
4	3.0	car_flow.1	0.00	164.78	2624.72	1.20	0.07	837.22	55.94

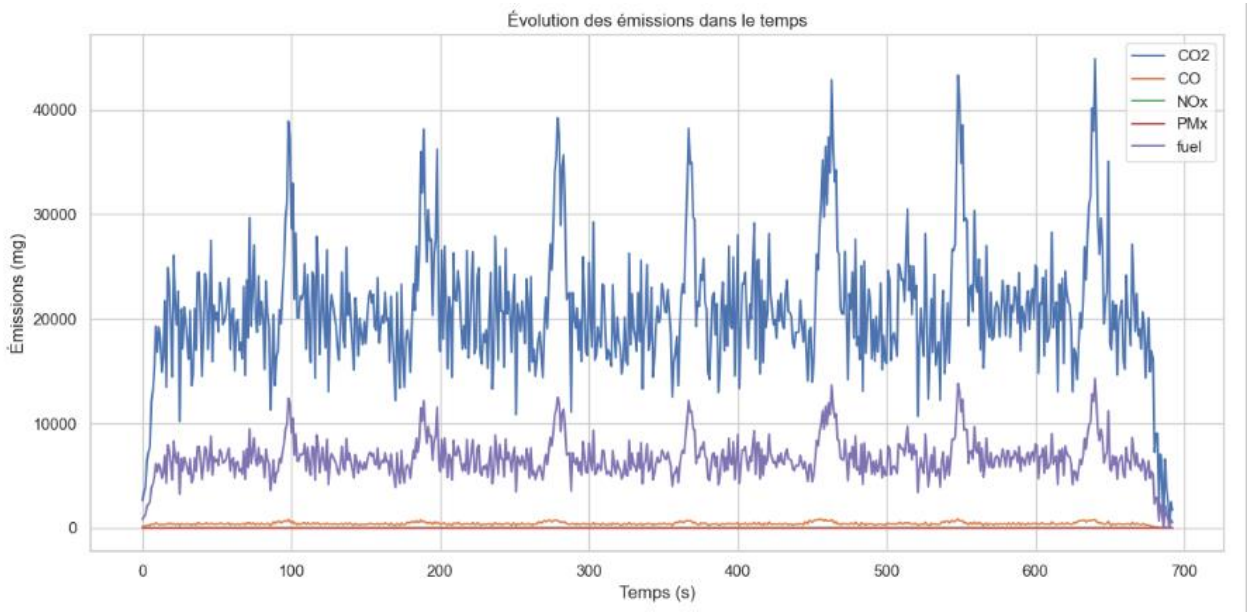
```
df_q = pd.DataFrame(data_q)
df_q.head()
```

	time_start	time	meanSpeed	haltingVehicles	waitingTime	occupancy
0	0.0	1.0	-1.00	0	0.0	0.0
1	1.0	2.0	2.00	0	0.0	0.0
2	2.0	3.0	3.69	0	0.0	0.0
3	3.0	4.0	4.98	0	0.0	0.0
4	4.0	5.0	3.82	0	0.0	0.0

Évolution des émissions dans le temps :

```
# Grouper par timestep pour avoir la somme totale à chaque seconde
df_time = df.groupby("time").sum().reset_index()

# 🟡 Courbe d'évolution de chaque polluant
polluants = ["CO2", "CO", "NOx", "PMx", "fuel"]
```

```
plt.figure(figsize=(12, 6))
for p in polluants:
    sns.lineplot(x="time", y=p, data=df_time, label=p)

plt.title("Évolution des émissions dans le temps")
plt.xlabel("Temps (s)")
plt.ylabel("Émissions (mg)")
plt.legend()
plt.tight_layout()
plt.show()
```



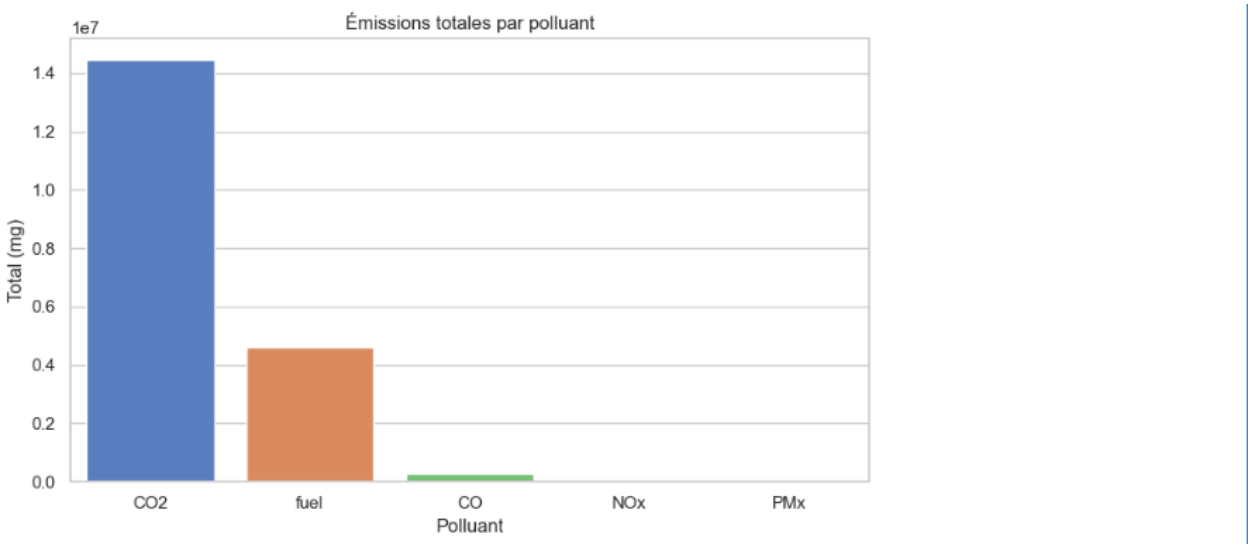
Ce graphique montre l'évolution des émissions de différents polluants ainsi que de la consommation de carburant, **au cours d'une simulation de trafic routier.**

Les pics sont **réguliers**, ce qui suggère un **cycle de circulation**, par exemple :

- des feux tricolores,
- des arrivées de vagues de véhicules,
- ou des changements de phase dans le trafic.

Emissions Totales par polluant :

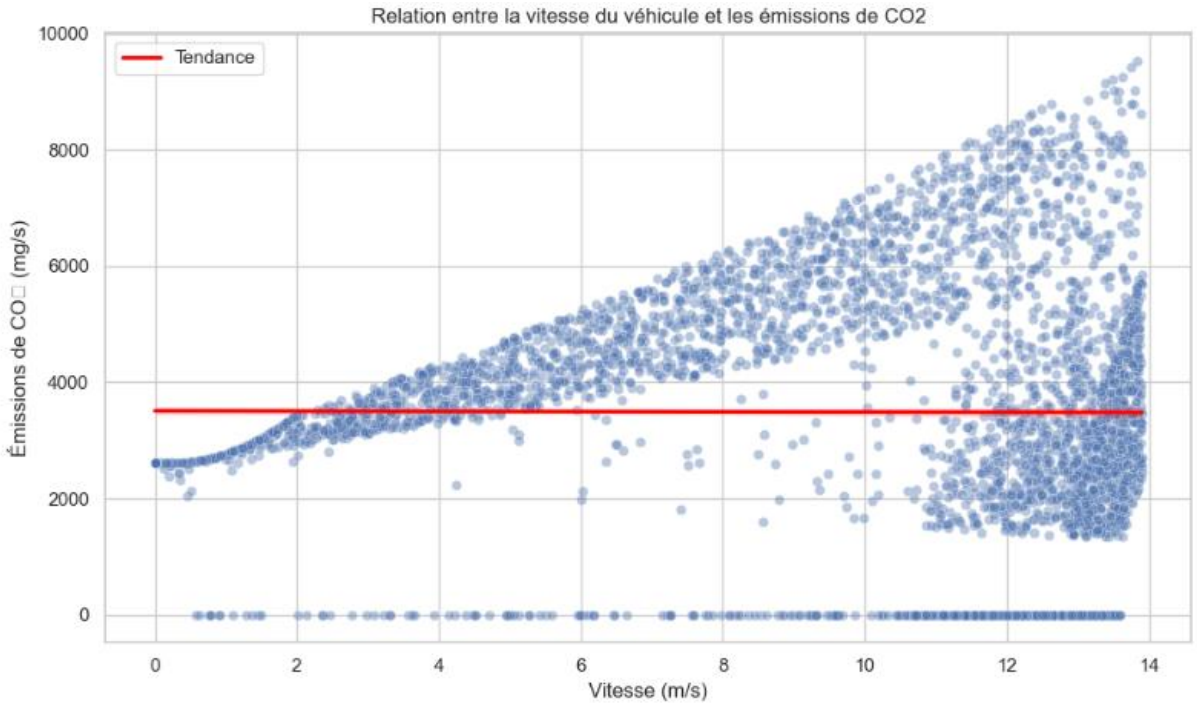
```
# 📊 Diagramme en barres des émissions totales
plt.figure(figsize=(8, 5))
total = df[polluants].sum().sort_values(ascending=False)
sns.barplot(x=total.index, y=total.values, hue=total.index, palette="muted", legend=False)
plt.title("Émissions totales par polluant")
plt.ylabel("Total (mg)")
plt.xlabel("Polluant")
plt.tight_layout()
plt.show()
```



Ce graphique révèle la répartition des émissions polluantes du trafic routier, où le CO2 domine (impact climatique), tandis que CO, NOx et PMx indiquent des problèmes locaux de combustion inefficace et de qualité de l'air, typiques des zones congestionnées. La présence significative de ces polluants secondaires suggère un besoin d'optimisation du trafic (feux synchronisés, réduction des embouteillages) et d'adoption de véhicules moins polluants.

Relation entre la vitesse du véhicule et les émissions de CO2 :

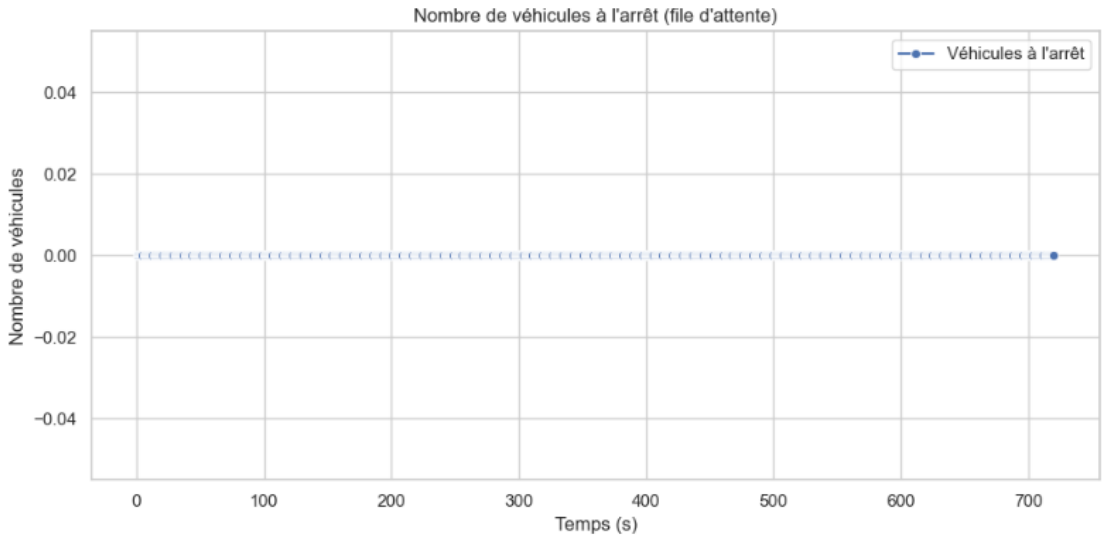
```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x="speed", y="CO2", alpha=0.4)
sns.regplot(data=df, x="speed", y="CO2", scatter=False, color='red', label="Tendance")

plt.title("Relation entre la vitesse du véhicule et les émissions de CO2")
plt.xlabel("Vitesse (m/s)")
plt.ylabel("Émissions de CO₂ (mg/s)")
plt.legend()
plt.tight_layout()
plt.show()
```



Le graphique montre la relation entre la vitesse d'un véhicule et ses émissions de CO_2, illustrée par un nuage de points et une courbe de tendance. Bien que les émissions aient tendance à augmenter légèrement avec la vitesse jusqu'à un certain point, la ligne de tendance relativement plate indique qu'il n'existe pas de corrélation linéaire forte entre les deux variables. On observe une grande dispersion des données, surtout à vitesses moyennes et élevées, ce qui suggère que d'autres facteurs comme l'accélération, la charge du véhicule ou le style de conduite influencent également les émissions. Ainsi, la vitesse seule n'explique pas de manière significative les variations d'émissions de CO_2.

Nombre de véhicules à l'arrêt (file d'attente) :

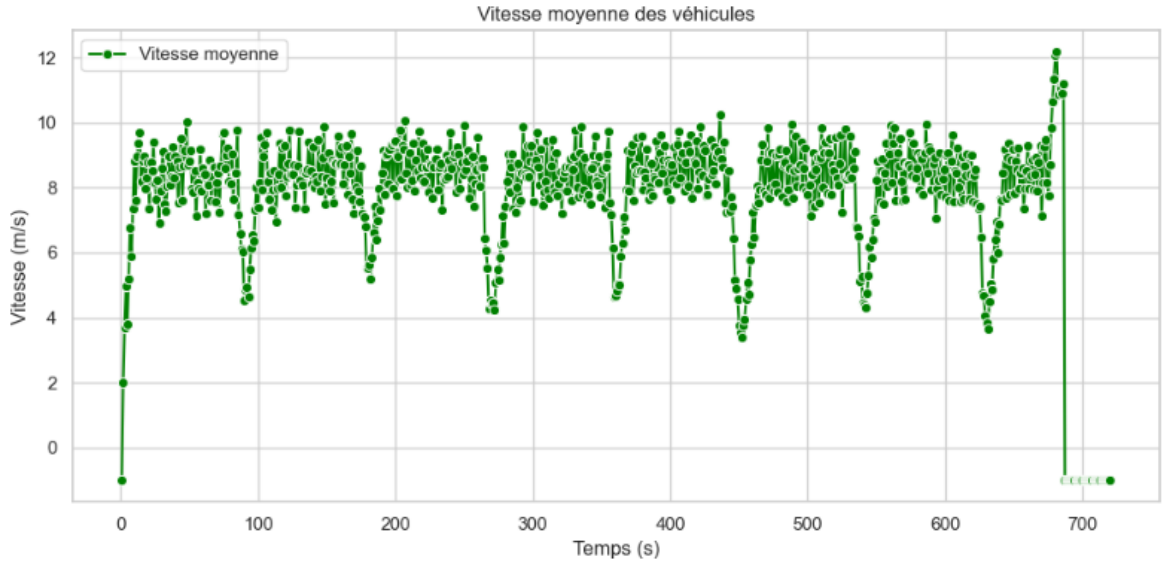
```
plt.figure(figsize=(10, 5))
sns.lineplot(data=df_q, x="time", y="haltingVehicles", marker="o", label="Véhicules à l'arrêt")
plt.title("Nombre de véhicules à l'arrêt (file d'attente)")
plt.xlabel("Temps (s)")
plt.ylabel("Nombre de véhicules")
plt.grid(True)
plt.tight_layout()
plt.show()
```



Le graphique montre l'évolution du nombre de véhicules à l'arrêt (file d'attente) en fonction du temps, sur une période d'environ 700 secondes. On observe que la courbe reste constamment à zéro, ce qui indique qu'à aucun moment un véhicule ne s'est retrouvé à l'arrêt. Cela suggère une circulation fluide et continue durant toute la période observée, sans congestion ni formation de file d'attente.

Vitesse moyens des véhicules :

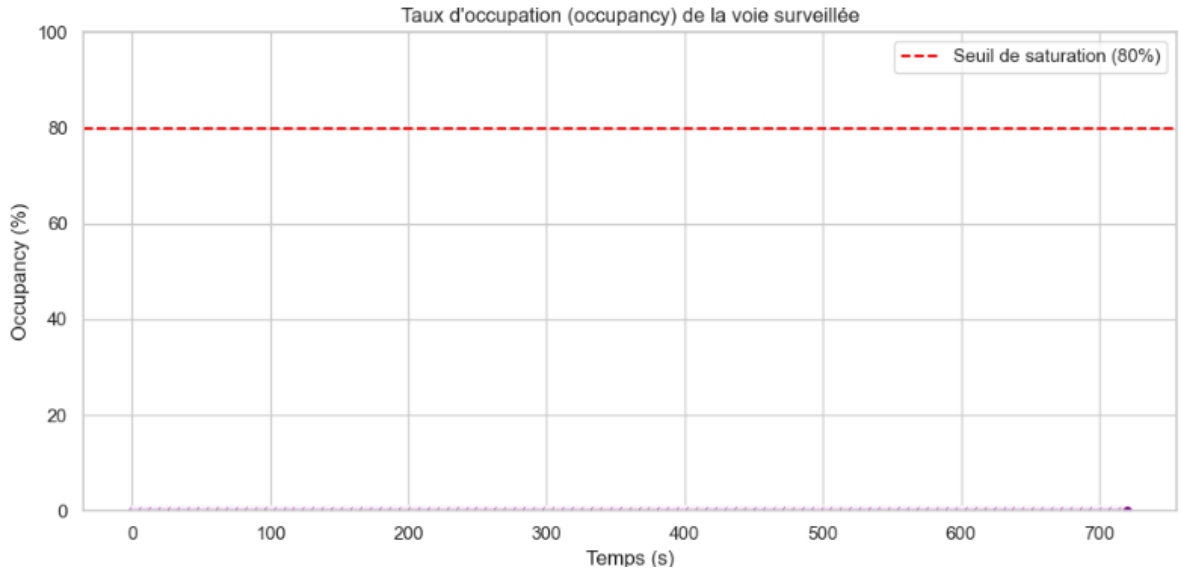
```
plt.figure(figsize=(10, 5))
sns.lineplot(data=df_q, x="time_end", y="meanSpeed", marker="o", color="green", label="Vitesse moyenne")
plt.title("Vitesse moyenne des véhicules")
plt.xlabel("Temps (s)")
plt.ylabel("Vitesse (m/s)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



Le graphique présente l'évolution de la vitesse moyenne des véhicules au fil du temps. On observe que la vitesse reste globalement élevée et stable, oscillant autour de 9 à 10 m/s, ce qui témoigne d'un trafic fluide. Cependant, on note des chutes ponctuelles de vitesse à des intervalles réguliers, suggérant des ralentissements temporaires, probablement dus à des obstacles ou des changements de conditions de circulation. Vers la fin de la période (autour de 700 s), on remarque une forte baisse soudaine de la vitesse jusqu'à zéro, ce qui pourrait indiquer un arrêt total du trafic ou la fin de la simulation.

Taux d'occupation de la voie surveillé :

```
df_q["occupancy_percent"] = df_q["occupancy"] * 100


plt.figure(figsize=(10, 5))
sns.lineplot(data=df_q, x="time", y="occupancy_percent", marker="o", color="purple")
plt.title("Taux d'occupation (occupancy) de la voie surveillée")
plt.xlabel("Temps (s)")
plt.ylabel("Occupancy (%)")
plt.ylim(0, 100)
plt.axhline(80, color='red', linestyle='--', label='Seuil de saturation (80%)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Le graphique illustre le taux d'occupation (occupancy) de la voie surveillée en pourcentage au cours du temps, avec une ligne rouge représentant le seuil de saturation fixé à 80 %. On constate que le taux d'occupation reste constamment très bas, proche de 0 %, durant toute la période observée, ce qui indique une circulation très fluide avec peu de véhicules présents simultanément sur la voie. Aucun dépassement du seuil critique n'est observé, ce qui suggère une infrastructure largement suffisante pour la demande de trafic enregistrée.

Contrôler les feux en temps réel :

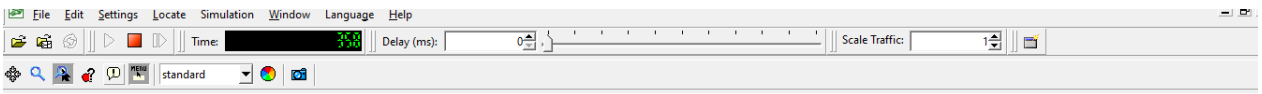
```
import traci
import sumolib
```

```python
import time


SUMO_BINARY = "sumo-gui"
CONFIG_FILE = "region.sumocfg"

# Lancer SUMO via TraCI
traci.start([SUMO_BINARY, "-c", CONFIG_FILE])

# ID du feu à contrôler
traffic_light_id = "n2"
detector_id = "queue_detector_edge1"

# Boucle de simulation
step = 0
while step < 1000:
    traci.simulationStep()

    # Lire la file d'attente
    queue_length = traci.lanearea.getLastStepHaltingNumber(detector_id)
    print(f"Step {step} - Halting vehicles: {queue_length}")

    # Si plus de 10 véhicules en attente, passer le feu au vert (phase 0)
    if queue_length > 10:
        traci.trafficlight.setPhase(traffic_light_id, 0)

    step += 1
    time.sleep(0.1)


traci.close()
```

Ce script :

- Lance SUMO
- Lit le nombre de véhicules arrêtés sur un détecteur (queue_detector_edge1)
- Passe le feu au vert si la file dépasse un seuil (par ex. 10 véhicules)

Exemple de simulation :



oading route-files incrementally from 'region.rou.xml'