# Modélisation et simulation du trafic routier

William AUROUX

Numéro d'inscription : 11264

# Plan

# Enjeux du trafic routier

- Selon l'INSEE, plus de 33 millions de véhicules en service dans l'hexagone en 2017.
- Dans un bouchon, la consommation d'essence au kilomètre est multipliée par deux. Une voiture pollue alors 200 fois plus que le train.
- Selon l'Inrix, en 2013, le coût des embouteillages urbains en France s'élèvent à 17 milliards d'euros. Les prédictions envisagent un coût de 21 milliards d'euros en 2030.
- Les embouteillages ont amputé en moyenne 136 heures à chaque conducteur en 2013.
- Selon une étude menée par l'entreprise Tomtom, en région parisienne, le trafic routier a produit 14 mégatonnes de CO2 en 2021.

# Problématique

Comment modéliser et simuler le trafic routier le plus fidèlement possible et comment fluidifier la circulation afin de limiter les congestions ?

# Plan

# Présentation

- Ce modèle, appelé modèle du conducteur intelligent est inventé en 2000 par Treiber, Hennecke et Helbing. Il a pour but d'exprimer l'accélération du i-ème véhicule avec les paramètres de ce véhicule et du véhicule précédent.
- En notant $x_i$ et $v_i$ la position et la vitesse du i-ème véhicule, le mouvement du véhicule i est donné par la formule :

$$\frac{dv_i}{dt} = a_{0i}\left(1 - \left(\frac{v_i}{v_{0,i}}\right)^{\delta} - \left(\frac{s^*(v_i, \Delta v_i}{s_i}\right)^2\right)$$

avec

$$s^*(v_i, \Delta v_i) = s_{0i} + v_i T_i + \frac{v_i \Delta v_i}{\sqrt{2 a_i b_i}}$$

On discrétise les expressions de $x_i(t)$ et $v_i(t)$ grâce à la formule de Taylor. Puis après mise en place de l'interface graphique avec les paramètres :

On peut dejà observer, après quelques secondes, la formation d'embouteillages.

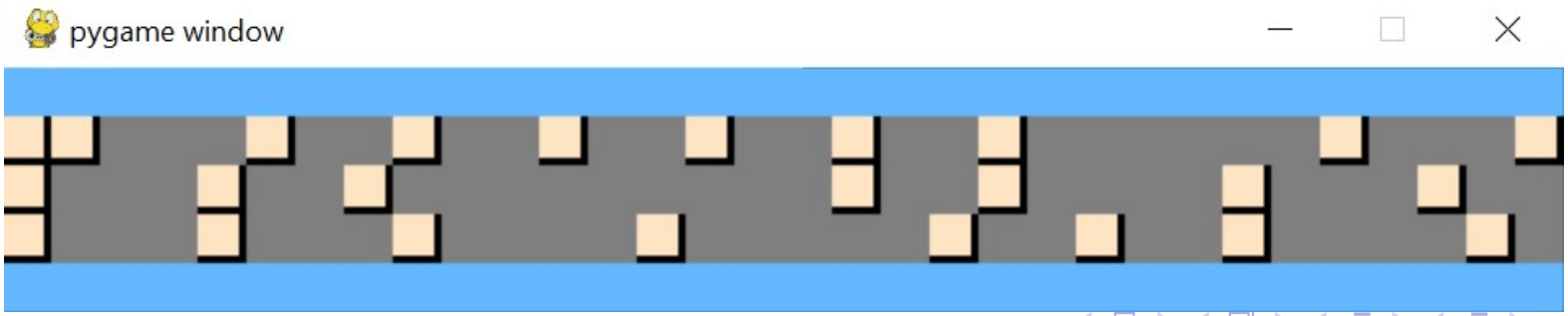
# Choix d'abandonner ce modèle

J'ai néanmoins pris la décision d'abandonner ce modèle car je rencontrais des difficultés à bien implémenter le changement de voie pour les véhicules.

# Plan

# Description de l'automate cellulaire

- Dans ce modèle : discrétisation du **temps** et de l'**espace**
- Les routes sont représentées par des tableaux et chaque véhicule prend la case de ce tableau correspondant à sa position sur la route.
- On fixe les paramètres des véhicules (les mêmes pour tous) qui modulent le déplacement de ces derniers : taille, vitesse maximale, probabilité de ralentissement, paramètres de freinages.
- À chaque instant t, on déplace les véhicules en fonction de la disponibilité des cases suivantes et de leur vitesse.



# Modèle de Nagel-Schreckenberg

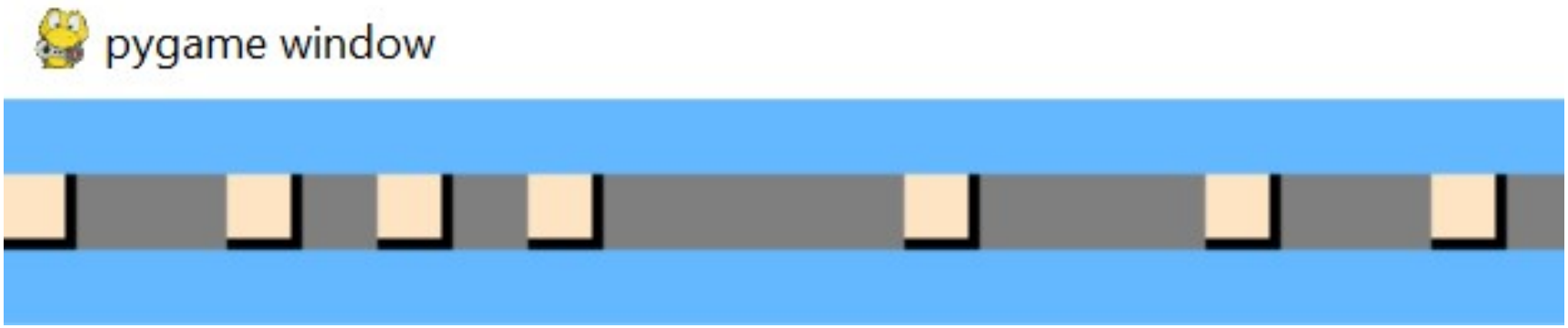
Dans ce modèle, la vitesse d'une voiture $v_n$ prend des valeurs entières de 0 à $v_{max}$. On note $x_n$ la position de la n-ième voiture, $d_n = x_{n+1} - x_n$ et f le terme de freinage (souvent égal à 1)

- Étape 1 : Changement de voie. Si le véhicule le peut, il change de voie avec une probabilité $p_{chgt}$ (même si le véhicule change de voie, il peut avancer sur sa nouvelle voie à cet instant).
- Étape 2 : Phase d'accélération. $v_n \leftarrow min(v_n + 1, v_{max})$
- Étape 3 : Phase de décélération si le véhicule est trop proche des véhicules alentours. Si $d_n \leq v_n$ : $v_n \leftarrow min(v_n, d_n - f)$
- Étape 4 : Mis en jeu d'un facteur aléatoire temporel. Si $v_n \geq 0$, alors $v_n \leftarrow min(v_n - f, 0)$ avec une probabilité p
- Étape 5 : Mise en mouvement. $x_n \leftarrow x_n + v_n$

# Interface graphique

Pour l'interface graphique, utilisation du module pygame dans Python. Simulation des voies par une matrice à coefficients dans [0,1,2]. Si le coefficient vaut :

- 0 : c'est un bout de route
- 1 : c'est un véhicule
- 2 : c'est l'arrière plan

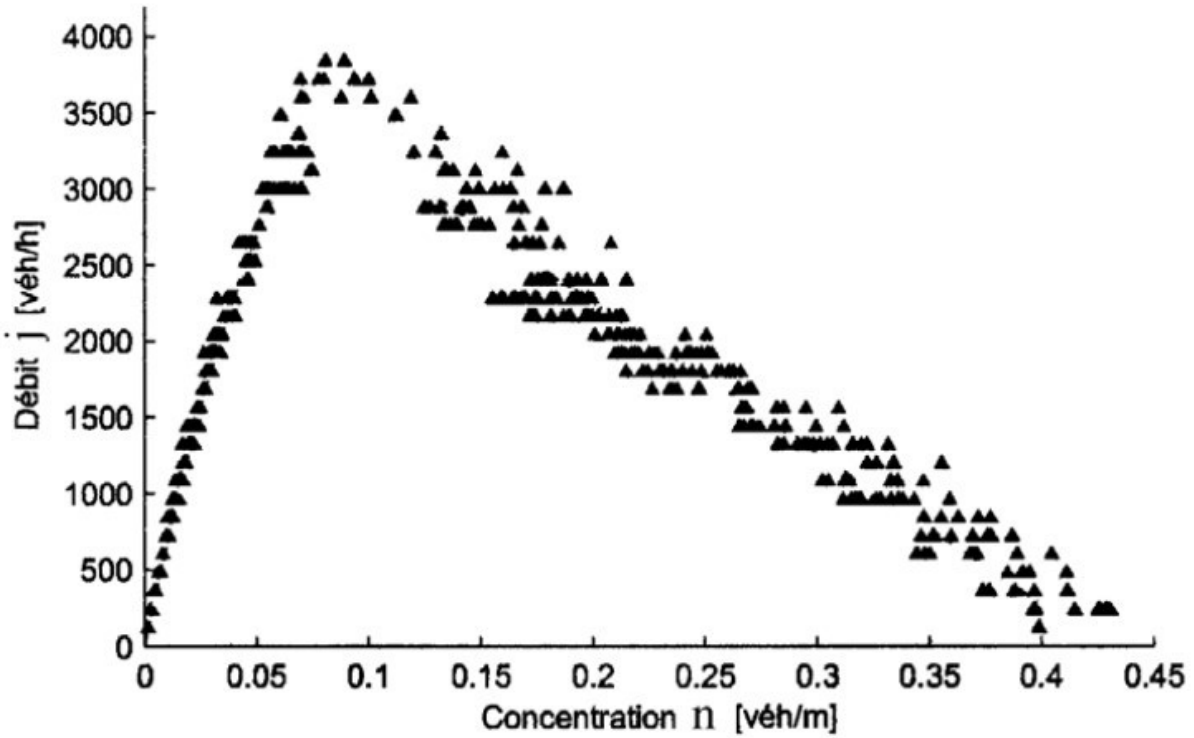


# Plan

# Diagramme fondamental du trafic

Le diagramme fondamental du trafic consiste à tracer la courbe du débit (en veh/h) par rapport à la concentration en véhicules sur la voie (en veh/m). Il permet de déterminer si la circulation est fluide ou congestionnée et caractérise cette fluidité.

Allure expérimentale réalisée en ville sur une seule voie sans obstacle



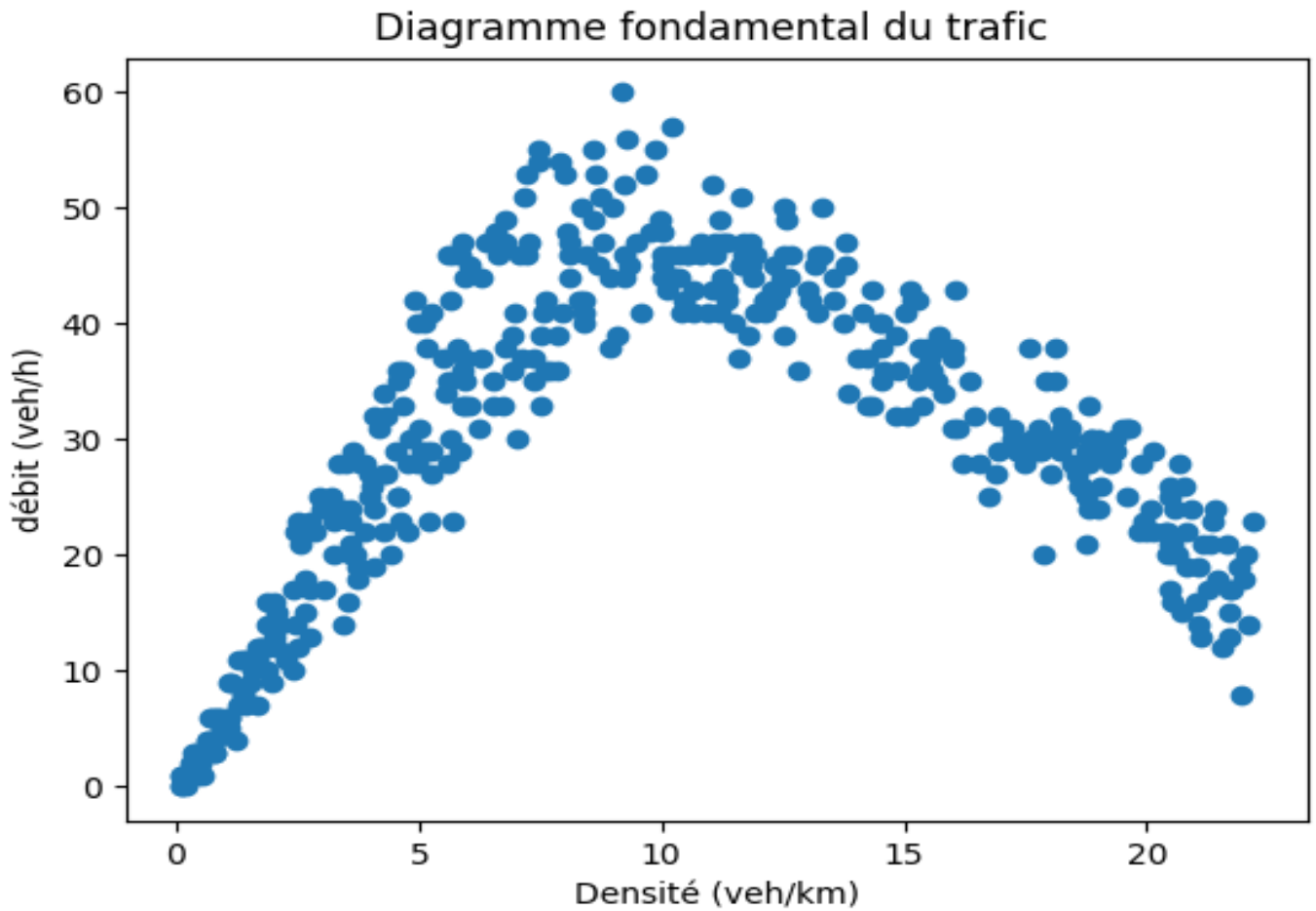

*Proposé dans le sujet de physique PSI du concours Centrale-Supélec en 2005 d'après la thèse de M.Leclercq*

# Obtention du diagramme

On pose 1h = 120 tours, 1km = 30 cases. On génère toutes les heures une carte avec un nombre aléatoire de voitures, positionnées aléatoirement sur la route. On mesure le débit pendant une heure en plaçant un point de mesure au milieu de la voie et on mesure la densité grâce au rapport nombre de véhicules sur taille de la voie.

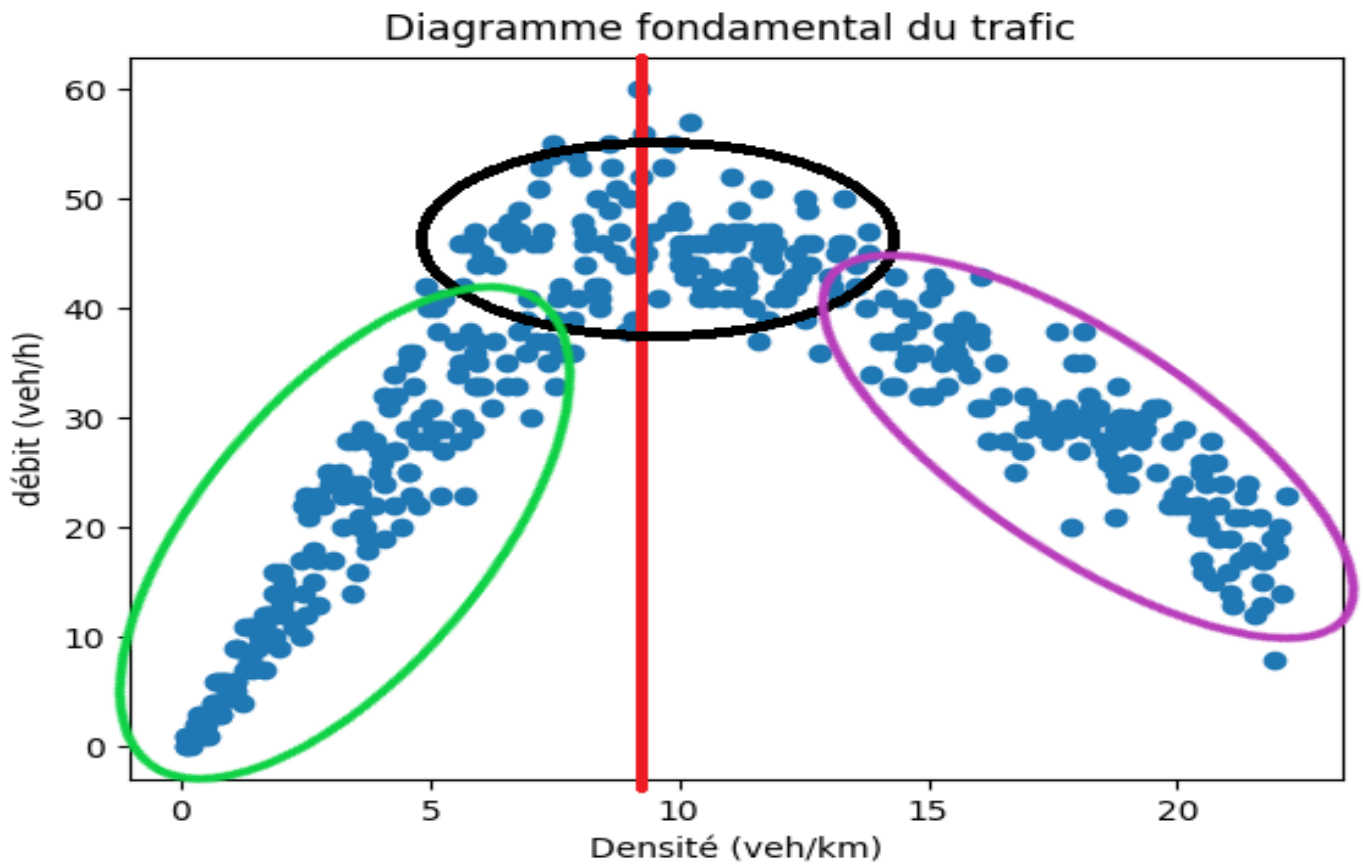
# Résultats

Voici le diagramme fondamental obtenu avec cette implémentation ( pour $v_{max} = 4, p = 0.1$ ). Il comporte 437 points.



# Analyse du diagramme

On observe que la densité critique de véhicules se trouve aux alentours de $n_c = 9$ veh/km pour un débit d'environ $q_{max} = 60$ veh/h, ce qui correspond à une vitesse $v_c \approx 1.66$ cases/temps.




vert = zone fluide ; noir = zone critique ; violet = zone congestionnée

# Plan

# Embouteillages fantômes

Les embouteillages urbains se forment parfois subitement, à la suite du freinage d'une seule voiture qui propage ce ralentissement de proche en proche. On observe alors une propagation de la congestion en « accordéon ».

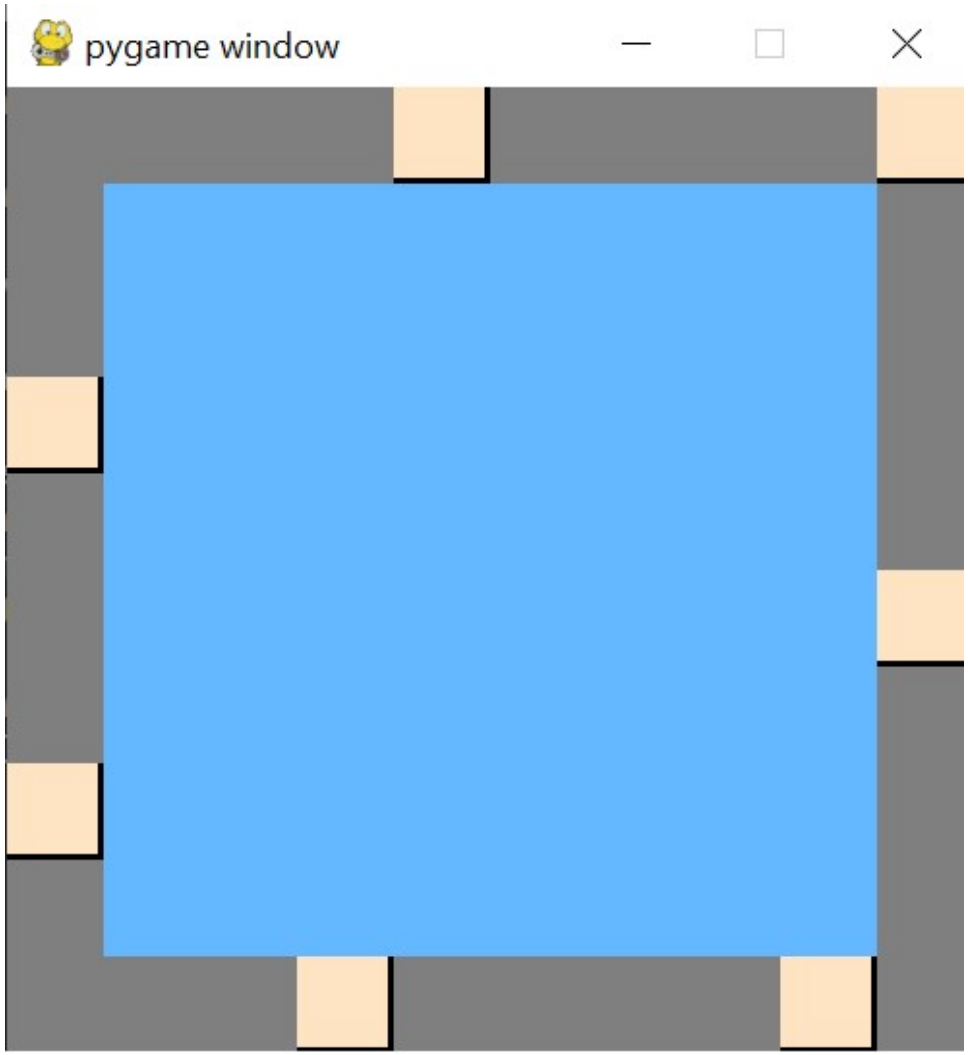
# Implémentaion

Pour simuler l'apparition d'un tel embouteillage :

- Les véhicules circulent sur un rond-point et on applique les mêmes règles que pour la simulation avec des voies horizontales.
- On fait s'arrêter complètement une voiture pendant un nombre de tours fini pour observer si la congestion apparaît.
- On s'intéresse à l'évolution des congestions en fonction du nombre de véhicules sur la boucle.

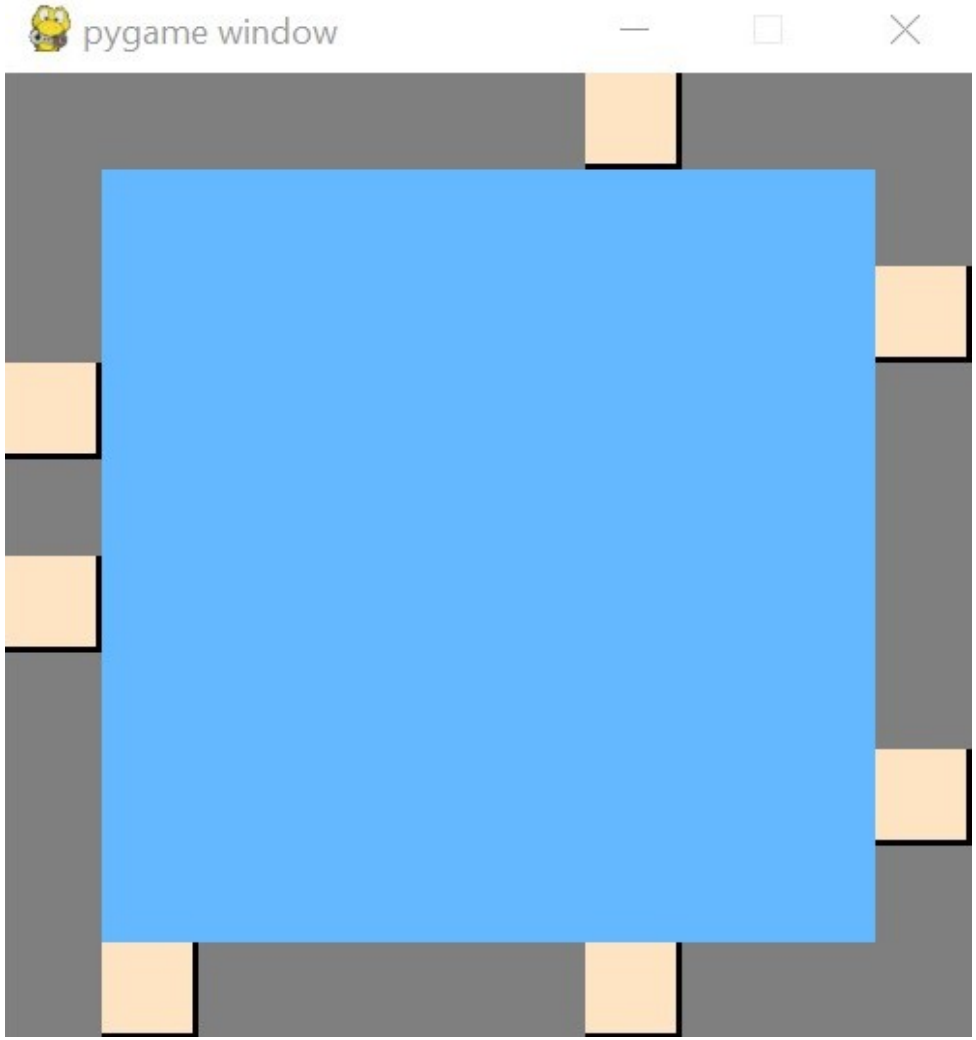
# Implémentaion

Pour $v_{max} = 2$ et un temps d'arrêt de 5 unités temporelles.



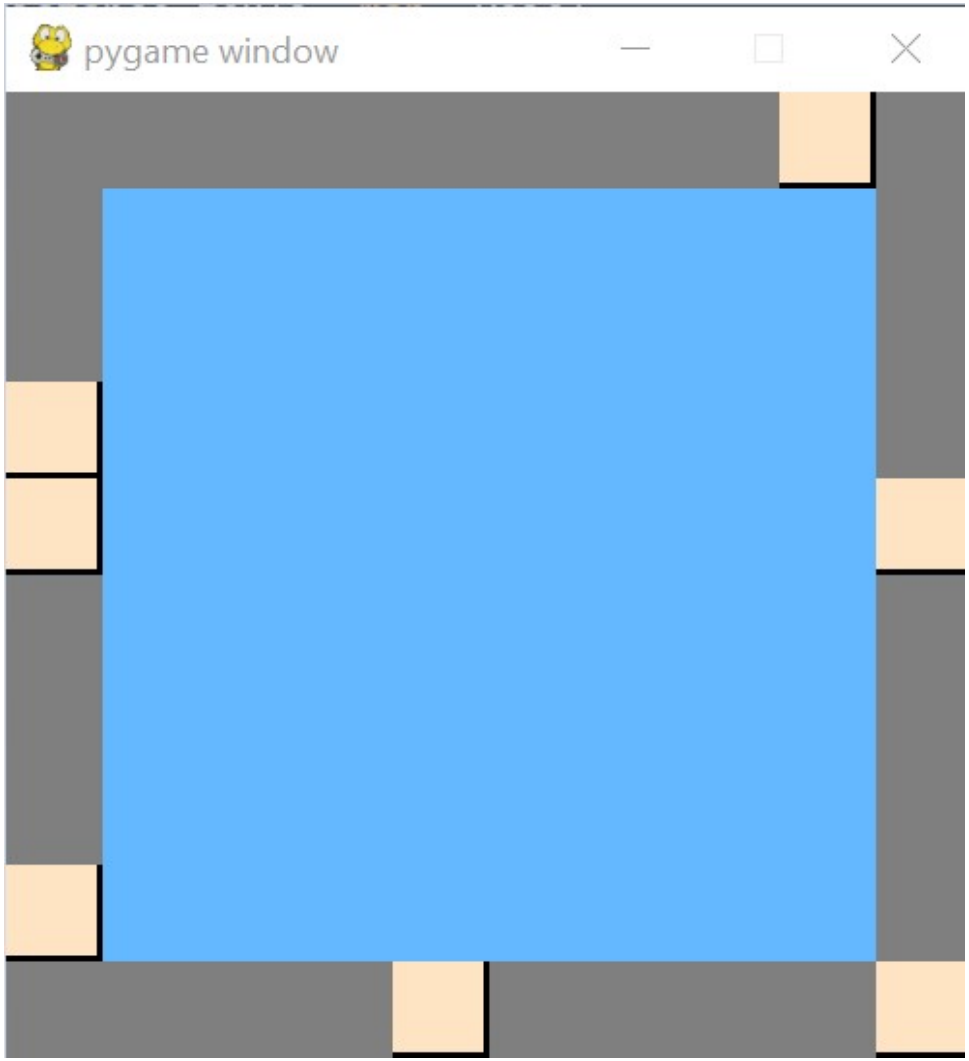
# Implémentaion

Pour $v_{max} = 2$ et un temps d'arrêt de 5 unités temporelles.



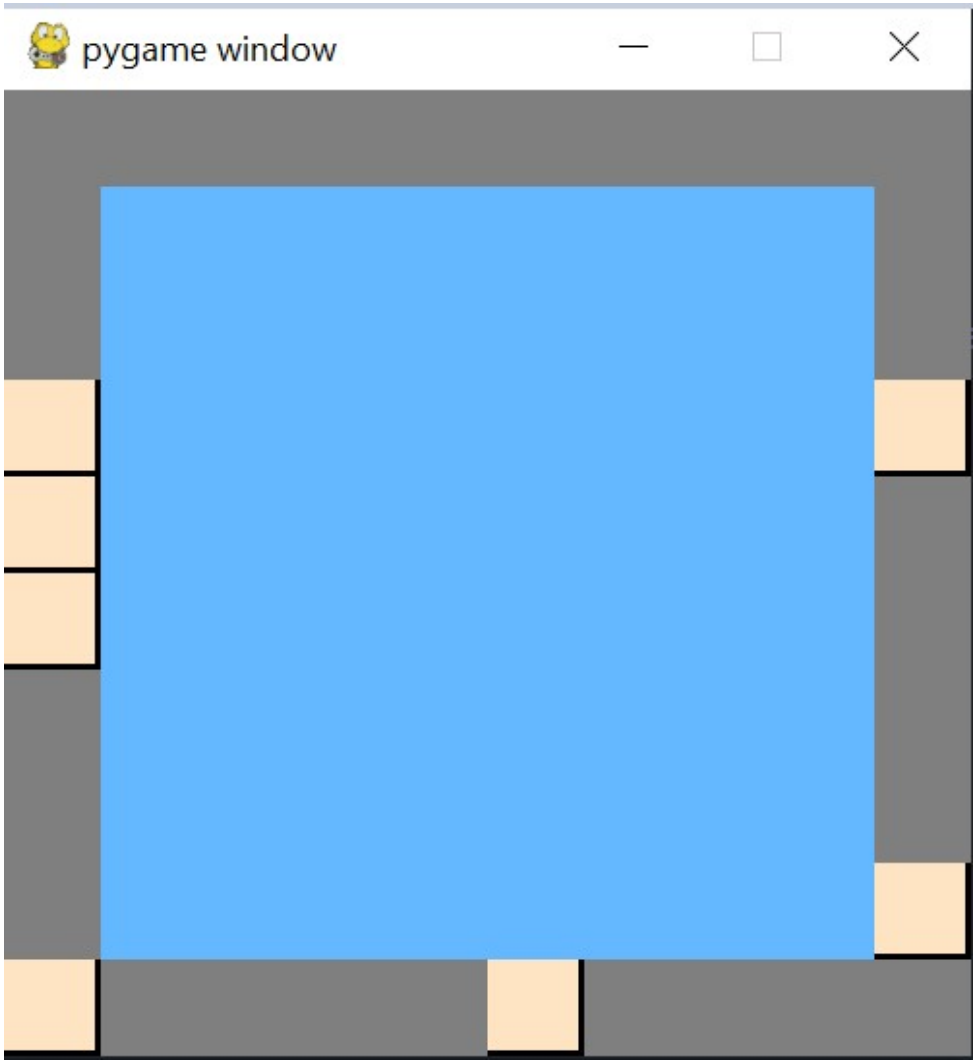
# Implémentaion

Pour $v_{max} = 2$ et un temps d'arrêt de 5 unités temporelles. -bn



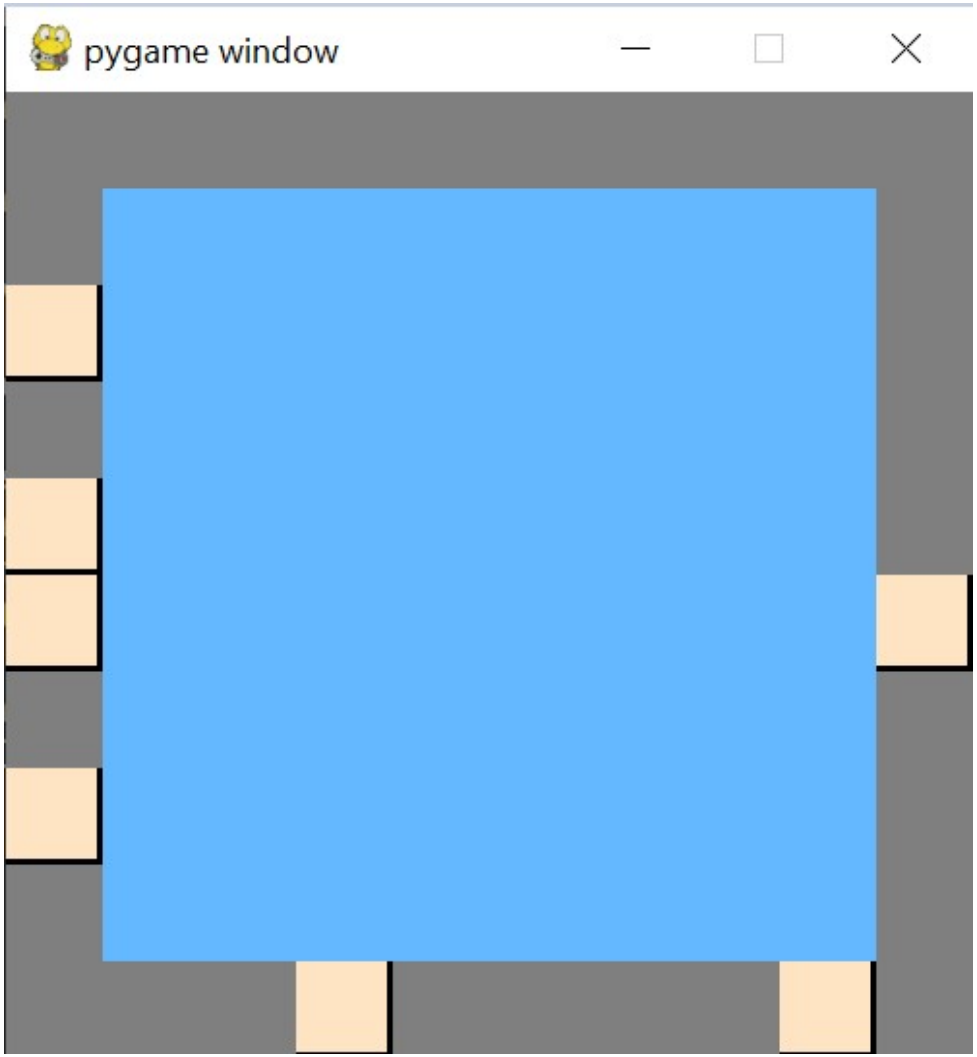
# Implémentaion

Pour $v_{max} = 2$ et un temps d'arrêt de 5 unités temporelles.



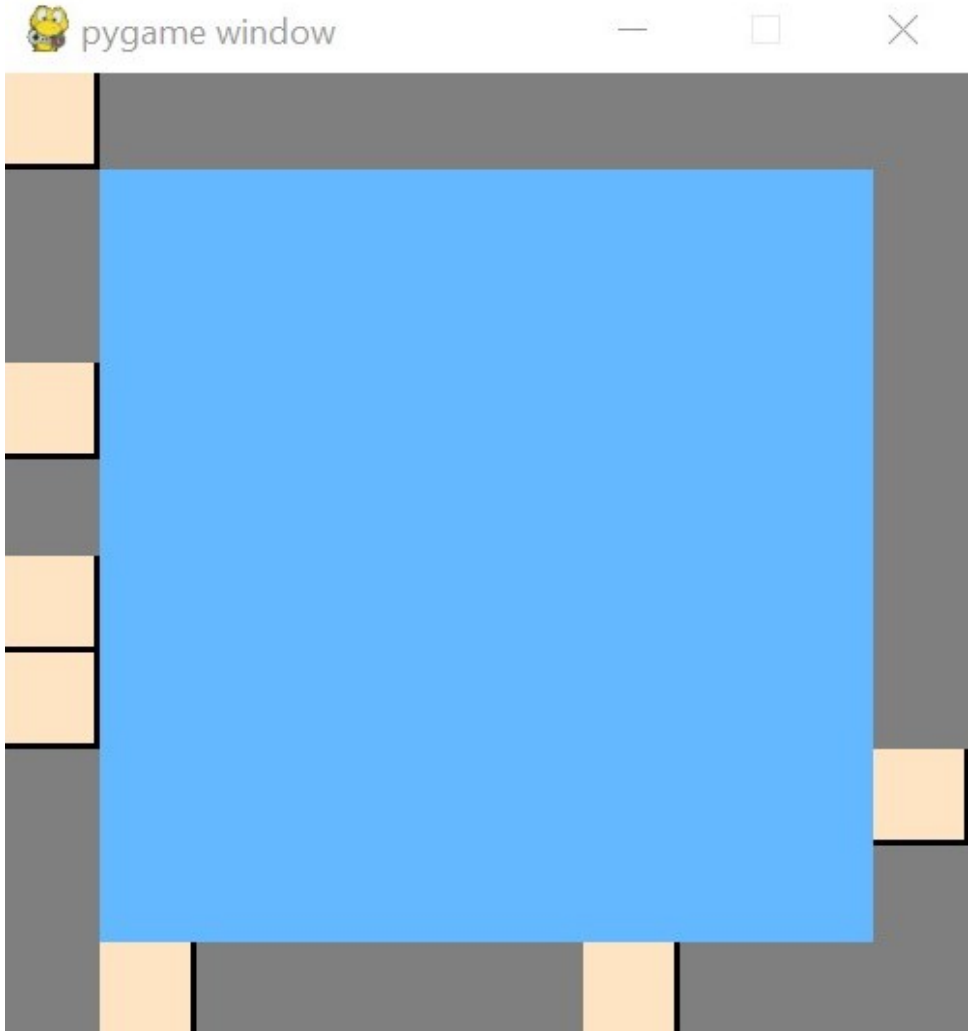
# Implémentaion

Pour $v_{max} = 2$ et un temps d'arrêt de 5 unités temporelles.



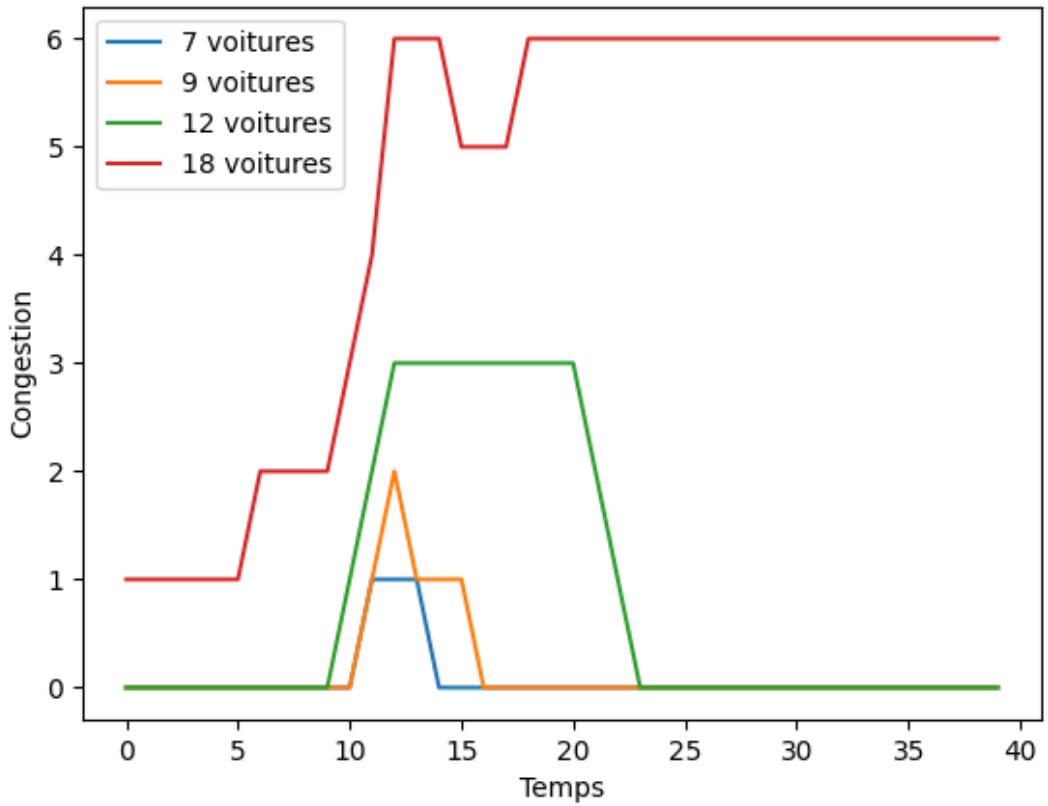
# Implémentaion

Pour $v_{max} = 2$ et un temps d'arrêt de 5 unités temporelles.



# Résultats

Avec $v_{max} = 4$ et un temps d'arrêt de 3 unités temporelles.



# Plan

# Conclusion

- On pourrait améliorer ce modèle en ajoutant de nouveaux paramètres comme des feux de freinage et une anticipation de la vitesse du véhicule qui précède la voiture (Break Light Model).
- On obtient alors une modélisation plus réaliste.
- Les automates cellulaires peuvent être utiles pour simuler le trafic routier avec un bon niveau de réalisme et pour proposer des solutions pour diminuer les congestions.

# Merci pour votre attention !

# Plan

# Rappel de la formule

$$\frac{dv_i}{dt} = a_{0i}\left(1 - \left(\frac{v_i}{v_{0,i}}\right)^{\delta} - \left(\frac{s^*(v_i, \Delta v_i}{s_i}\right)^2\right)$$

avec

$$s^*(v_i, \Delta v_i) = s_{0i} + v_i T_i + \frac{v_i \Delta v_i}{\sqrt{2 a_i b_i}}$$

# Paramètres

Avec :

- $s_{0i}$ : distance minimale désirée entre les véhicules i et i-1.
- $v_{0i}$ : vitesse maximale désirée du véhicule i.
- $\delta$ : facteur exponentiel. Il contrôle la "continuité" de la variation de l'accélération.
- $T_i$ : temps de réaction du conducteur de la voiture i.
- $a_{0i}$ : accélération maximale du véhicule i.
- $b_i$ : décélération voulue pour la voiture i
- $s^*$ : distance désirée entre 2 véhicules consécutifs.
- $s_i$ : distance entre le véhicule i et i-1.
- $v_i T_i$ : distance parcourue par le véhicule avant que le conducteur réagisse.
- $\frac{v_i \Delta v_i}{\sqrt{2 a_i b_i}}$ : distance de freinage à partir du moment où le conducteur réagit.

# Code

```
from numpy import *
from random import *
from pygame import *
from math import *
from time import *
from matplotlib import *
import pylab as py
import pygame


def average(l):
    sum = 0
    for k in range(len(l)):
        sum += l[k]
    return (sum / len(l))


def car_position(i, nb_voie, carte):
    """
    Retourne la position de la voiture sur la voie
    """
    map = carte.copy()
    j = nb_voie
    cmpt, x = 0, 0
    while cmpt != i:
        if map[j, x] == 1:
            cmpt += 1
        if cmpt != i:
            x += 1
        if x == map.shape[1]:
            return ("pas de voiture", i, " dans cette voie")
    return x


def pos_end_road(carte, voie):
    return len(carte[voie] - 1)
```

```
def can_move(i, v, nb_voie, carte):
    """
    indique si la voiture peut move de v cases
    """
    map = carte.copy()
    n = nb_voie
    x = car_position(i, n, map)

    if 1 not in [map[n, p] for p in range(x + 1, x + v + 1)] :
        return True
    else:
        return False

def chgt_voie_possible(i, v, voie, carte, h_b):
    """
    retourne True si le changement de voie est possible
    """
    if nb_voiture(carte, voie) == 1 and car_position(i, voie, carte) in [len(carte[0]) - k for k in
                                                                         range(0, v)]:
        return False
    if h_b == 'b':
        x = car_position(i, voie, carte)
        if carte[voie + 1, x] == 2:
            return False
        if end_road(i, v, carte, voie):
            return False
        if 1 not in [carte[voie + 1, p] for p in range(x + 1, x + v + 1)] and 5 not in [carte[voie + 1, p] for p in
                                                                                        range(x + 1, x + v + 1)]:
            return True
        return False

    if h_b == 'h':
        x = car_postion(i, voie, carte)
        if carte[voie - 1, x] == 2:
            return False
        if end_road(i, v, carte, voie):
            return False
        if 1 not in [carte[voie - 1, p] for p in range(x + 1, x + v + 1)] and 5 not in [carte[voie - 1, p] for p in
                                                                                        range(x + 1, x + v + 1)]:
            return True
        return False
```

```python
def chgt_voie(i, v, voie, carte, h_b):
    """
    La voiture i change voie de toujours une case en diagonale en haut ou en bas
    """
    x = car_position(i, voie, carte)
    if h_b == 'b':
        if carte[voie + 1, x + 1] == 1:
            return carte
        carte[voie, x] = 0
        carte[voie + 1, x + 1] = 1
    if h_b == 'h':
        if carte[voie - 1, x + 1] == 1:
            return carte
        carte[voie, x] = 0
        carte[voie - 1, x + 1] = 1
    return carte


def move(i, v, nb_voie, carte):
    """
    fait move la voiture i
    """
    if not can_move(i, v, nb_voie, carte):
        return ("la voiture ne peut pas move")
    j = nb_voie
    if can_move(i, v, nb_voie, carte):
        x = car_position(i, nb_voie, carte)
        carte[j, x] = 0
        carte[j, x + v] = 1
    return carte
```

```python
def end_road(i, v, carte, nb_voie):
    """
    Renvoie True si la voiture est entre la fin de la route et fin de la route - v
    Renvoie False si non
    """
    fin = carte.shape[1]
    x = car_position(i, nb_voie, carte)
    if x in [s for s in range(fin - v, fin + 1)]:
        return True
    return False


def nb_voiture(carte, voie):
    """indique le nombre de voitures présentes sur la voie """
    cmpt = 0
    for i in range(carte.shape[1]):
        if carte[voie, i] == 1:
            cmpt += 1
    return cmpt


def add_voiture(carte, voie):
    carte[voie, 0] = 1
    return carte



def distance_v(i, carte, voie):
    """
    renvoie la distance entre la voiture i et la voiture i+1
    """
    pos1 = car_position(i, voie, carte)
    if i != nb_voiture(carte, voie):
        pos2 = car_position(i + 1, voie, carte)
        return (pos2 - pos1 - 1)
    else:
        return pos_end_road(carte, voie) - pos1
```

```
def pliage(Carte):
    """
    permet de créer une carte en forme de boucle
    """
    M = ones((10, 10)) * 2
    M[0, 0] = Carte[0, 0]
    for j in range(1, 10):
        M[0, j] = Carte[0, j]
    for i in range(1, 10):
        M[i, 9] = Carte[0, i + 9]
    for j in range(1, 10):
        M[9, -j] = Carte[0, j + 17]
    for i in range(1, 10):
        M[-i, 0] = Carte[0, i + 26]
    return M


def compteur(Carte, voie):
    """
    fonction comptant le nombre total de voitures collées, permet de quantifier la congestion
    """
    cmpt = 0
    for i in range(0, Carte.shape[1] - 1):
        if Carte[voie, i] == 1 and Carte[voie, i + 1] == 1:
            cmpt += 1
    return cmpt
```

```
def circulationunevoie_hori(carte, v, voie, pro, ral):
    """
    simulation de la circulation à une voie. pro est la probabilité d'apparition d'une voiture en début de route,
    ral la probabilité qu'une voiture ralentissent
    """
    assert (v > 1)
    map = carte.copy()
    n, p = map.shape
    taille_carre_pixel = 10
    init()
    window = display.set_mode((p * taille_carre_pixel, n * taille_carre_pixel))
    Running = True
    N = nb_voiture(map, voie)
    vitesse = [0 for _ in range(N)]
    while Running:
        N = nb_voiture(map, voie)
        for i in range(1, N + 1):
            coordonné_voiture_i = (voie, car_position(i, voie, map))
            v_i = vitesse[i-1]
            v_i = min(v_i + 1, v)
            if distance_v(i,map,voie) <= v_i:
                v_i = min(v,distance_v(i,map,voie) - 1)
            pral = random()
            if pral > ral:  # cas où la voiture ne ralentis pas
                if end_road(i, v_i, map, voie):
                    map[voie, car_position(i, voie, map)] = 0
                    vitesse[i-1:i] = vitesse[i-1:i-1]
                else:
                    if can_move(i, v_i, voie, map) :
                        move(i, v_i, voie, map)
                        vitesse[i-1] = v_i
```

```
            else:  # cas où la voiture ralentis
                if end_road(i, v_i, map, voie):
                    map[voie, car_position(i, voie, map)] = 0
                    vitesse[i -1:i] = vitesse[i -1:i -1]
                else:
                    if can_move(i, v_i, voie, map) :
                        move(i, v_i - 1, voie, map)
                        vitesse[i - 1] = v_i

        pi = random()
        if pi < pro and map[voie, v] != 1:
            add_voiture(map, voie)
            vitesse.append(0)

        fond = afficher(map)
        window.blit(fond, (0, 0))
        display.flip()
        for event in pygame.event.get():
            if event.type == QUIT:
                Running = False
        sleep(0.35)
    quit()
```

```
def circulation_k_voies(carte, v, ListeVoie, pro_ral, p_chgt_voie):
    """
    simulation de circulation pour plusieurs voies
    """
    map = carte.copy()
    running = True
    n, p = map.shape
    taille_carre_pixel = 60
    init()
    window = display.set_mode((p * taille_carre_pixel, n * taille_carre_pixel))
    mat_vitesse = [[0 for _ in range(nb_voiture(map,ListeVoie[k]))] for k in range(len(ListeVoie))]
    while running:
        # changement de voie
        for k in range(len(ListeVoie)):
            voie = ListeVoie[k]
            if nb_voiture(map, voie) != 0:
                for i in range(1, nb_voiture(map, voie) - 1):
                    p = random()
                    if p < p_chgt_voie:
                        p = randint(0, 1)
                        if p == 0:
                            if chgt_voie_possible(i, v, voie, map, 'h'):
                                chgt_voie(i, v, voie, map, 'h')
                        if p == 1:
                            if chgt_voie_possible(i, v, voie, map, 'b'):
                                chgt_voie(i, v, voie, map, 'b')
```

```
        # les voitures avance comme pour une circulation une voie
        for k in range(len(ListeVoie)):
            voie = ListeVoie[k]
            vitesse_k = mat_vitesse[k]
            tic_circulation_une_voie(map, v, voie, 0.8, pro_ral,vitesse_k)

        fond = afficher(map)

        window.blit(fond, (0, 0))
        display.flip()
        for event in pygame.event.get():
            if event.type == QUIT:
                running = False
        sleep(0.3)
        """
    quit()
```

```
def tic_circulation_une_voie(carte, v, voie, pro, ral,vitesse):
    """
    change la carte en simulant un seul passage dans la boucle while de la circulation une voie
    """
    map = carte
    N = nb_voiture(map, voie)
    for i in range(1, N + 1):
        coordonné_voiture_i = (voie, car_position(i, voie, map))
        v_i = vitesse[i - 1]
        v_i = min(v_i + 1, v)
        if distance_v(i, map, voie) <= v_i:
            v_i = min(v, distance_v(i, map, voie) - 1)
        pral = random()
        if pral > ral:  # cas où la voiture ne ralentis pas
            if end_road(i, v_i, map, voie):
                map[voie, car_position(i, voie, map)] = 0
                vitesse[i - 1:i] = vitesse[i - 1:i - 1]
            else:
                if can_move(i, v_i, voie, map):
                    move(i, v_i, voie, map)
                    vitesse[i - 1] = v_i

        else:  # cas où la voiture ralentis
            if end_road(i, v_i, map, voie):
                map[voie, car_position(i, voie, map)] = 0
                vitesse[i - 1:i] = vitesse[i - 1:i - 1]
            else:
                if can_move(i, v_i, voie, map):
                    move(i, v_i - 1, voie, map)
                    vitesse[i - 1] = v_i

    pi = random()
    if pi < pro and map[voie, v] != 1:
        add_voiture(map, voie)
        vitesse.append(0)

    return map
```

```
def circulation_fantome(carte, v, voie, temps_arret, timing, tours):
    """
    circulation sur une boucle , une voiture s'arrête alors pendant temps_arret tours pour essayer d'observer l'apparition d'embouteillage
    """
    assert (v > 1)
    Temps = []
    Congestion = []
    Temps_arret = []
    temps = 0
    pos_voit_arret = -1
    cmpt = 0
    map = carte.copy()
    n, p = map.shape
    taille_carre_pixel = 60
    init()
    window = display.set_mode((p * taille_carre_pixel, n * taille_carre_pixel))
    Running = True
    vitesse = [0 for _ in range(nb_voiture(map,voie))]
```

```
while Running:
    N = nb_voiture(map, voie)
    for i in range(1, N + 1):
        v_i = vitesse[i - 1]
        v_i = min(v_i + 1, v)
        if distance_v(i, map, voie) <= v_i:
            v_i = min(v, distance_v(i, map, voie) - 1)
        pral = random()
        coordonné_voiture_i = (voie, car_position(i, voie, map))
        if temps == timing and i == 7:  # on voudra arreter la voiture 7 pendant 'temps_arret' tours pour créer un bouchon fantôme
            pos_voit_arret = car_position(i, voie, map)
        if temps >= timing and car_position(i, voie, map) == pos_voit_arret and cmpt < temps_arret:
            cmpt += 1
        else:
            if end_road(i, v_i, map, voie):
                y = map.shape[1] - car_position(i, voie, map)
                if i == N:
                    if 1 not in [map[voie, l] for l in range(0, v_i)]:
                        map[voie, car_position(i, voie, map)] = 0
                        map[voie, v_i - y] = 1
                        vitesse[i-1] = v_i
            else:
                if can_move(i, v_i, voie, map):
                    move(i, v_i, voie, map)
                    vitesse[i - 1] = v_i
```

```
            Temps.append(temps)
            Congestion.append(compteur(map, voie))
            Temps_arret.append(timing)
            fond = afficher(pliage(map))
            window.blit(fond, (0, 0))
            display.flip()
            for event in pygame.event.get():
                if event.type == QUIT:
                    Running = False
            sleep(0.20)
            temps += 1
            if temps == tours:
                Running = False
    quit()
    return (Temps, Congestion)
```

```
def experience_fantôme(ListeCarte,v, voie, temps_arret, timing, tours):
    X = []
    Y = []
    for k in range(0, len(ListeCarte)):
        Carte = ListeCarte[k]
        XY = circulation_fantome(Carte, v, voie, temps_arret, timing, tours)
        X.append(XY[0])
        Y.append(XY[1])
    for i in range(0, len(X)):
        py.plot(X[i], Y[i], label=str(nb_voiture(ListeCarte[i], voie)) + ' ' + "voitures")
        py.legend()
    py.xlabel("Temps")
    py.ylabel("Congestion")
    py.show()
```

```
def diagramme_fondamental(carte, voie, v, ral, pro, pos_mesure_débit):

    assert (v > 1)
    debit = []
    densite = []
    den = []
    temps = 0
    tours = 0
    débit = 0
    nb_pt = 0
    map = carte.copy()
    map[voie - 1, pos_mesure_débit] = 6
    map[voie + 1, pos_mesure_débit] = 6
    n, p = map.shape
    taille_carre_pixel = 60
    init()
    window = display.set_mode((p * taille_carre_pixel, n * taille_carre_pixel))
    Running = True
    vitesse = [0 for _ in range(N)]

    # tous les 60 temps, pro augmente de 1
    # on pose 1h =120 tours et  1 km =30 cases
```

```
while Running:
    if temps == 120:
        nbr_voiture = randint(0, 250)
        map = array([[2 for k in range(0, 300)],
                     [0 for k in range(0, 300)],
                     [2 for k in range(0, 300)]])
        map[voie - 1, pos_mesure_débit] = 6
        map[voie + 1, pos_mesure_débit] = 6
        for k in range(0, nbr_voit):
            pos = randint(0, 295)
            while map[voie, pos] != 0:
                pos = randint(0, 295)
            map[voie, pos] = 1
        temps = 0
        vitesse = [randint(0,v) for _ in range(N)]

    N = nb_voiture(map, voie)

    for i in range(1, N + 1):
        coordonné_voiture_i = (voie, car_position(i, voie, map))
        v_i = vitesse[i - 1]
        v_i = min(v_i + 1, v)
        if distance_v(i, map, voie) <= v_i:
            v_i = min(v, distance_v(i, map, voie) - 1)
        pral = random()
        if pral > ral:  # cas où la voiture ne ralentis pas
            if end_road(i, v_i, map, voie):
                map[voie, car_position(i, voie, map)] = 0
                vitesse[i - 1:i] = vitesse[i - 1:i - 1]
            else:
                if can_move(i, v_i, voie, map):
                    move(i, v_i, voie, map)
                    vitesse[i - 1] = v_i
```

```
    else:  # cas où la voiture ralentis
        if end_road(i, v_i, map, voie):
            map[voie, car_position(i, voie, map)] = 0
            vitesse[i - 1:i] = vitesse[i - 1:i - 1]
        else:
            if can_move(i, v_i, voie, map):
                move(i, v_i - 1, voie, map)
                vitesse[i - 1] = v_i

    if pos_mesure_débit > coordonné_voiture_i[1] and pos_mesure_débit <= car_position(i, voie,map):
        # dans ce cas la voiture sera passé par la case où l'on mesure le débit
        débit += 1
pi = random()
if pi < pro and map[voie, v] != 1:
    add_voiture(map, voie)
    vitesse.append(0)


den.append(densite(map, voie))
if tours == 120:
    debit.append(débit)
    débit = 0
    densite.append(average(den))
    den = []
    tours = 0
    nb_pt += 1
```

```
            temps += 1
            tours += 1
            fond = afficher(map)
            window.blit(fond, (0, 0))
            display.flip()
            for event in pygame.event.get():
                if event.type == QUIT:
                    Running = False
            sleep(0.01)
        quit()
        py.scatter(densite, debit)
        py.title("Diagramme fondamental du trafic ")
        py.xlabel("Densité (veh/km)")
        py.ylabel("Débit (veh/h)")
        py.show()
        print("Nombre de points :", nb_pt)


def densite(carte, voie):
    """
    retourne la densité du trafic en veh/km
    """
    # on choisit 1km=30 cases
    assert (pos_end_road(carte, voie) > 30)
    n = nb_voiture(carte, voie)
    return ((n * 30) / pos_end_road(carte, voie))
```

```
def afficher(M):
    """
    permet l'affichage graphique en utilisant le module pygame
    """
    n, p = M.shape
    taille_carre_pixel = 8
    init()  # initialise la fenêtre
    window = display.set_mode((p * taille_carre_pixel, n * taille_carre_pixel))
    fond = Surface((p * taille_carre_pixel, n * taille_carre_pixel))
    fond.fill((22, 184, 78))
    for i in range(n):
        for j in range(p):

            if M[i, j] == 0:
                draw.rect(fond, (127, 127, 127), [j * taille_carre_pixel, i * taille_carre_pixel, taille_carre_pixel, taille_carre_pixel])

            if M[i, j] == 1:
                ""
                draw.rect(fond, (0, 0, 0), [j * taille_carre_pixel, i * taille_carre_pixel, taille_carre_pixel, taille_carre_pixel])
                ""
                draw.rect(fond, (255, 228, 196),
                          [j * taille_carre_pixel, i * taille_carre_pixel, taille_carre_pixel - 3, taille_carre_pixel - 3])

            if M[i, j] == 6:
                ""
                draw.rect(fond, (0, 0, 0), [j * taille_carre_pixel, i * taille_carre_pixel, taille_carre_pixel, taille_carre_pixel])
                ""
                draw.rect(fond, (255, 255, 0), [j * taille_carre_pixel, i * taille_carre_pixel, taille_carre_pixel - 3, taille_carre_pixel - 3])

    return fond
```

```
Map_1voie = array([[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                    2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
                   [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                   [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                    2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]])

Map_3voies = array(
    [[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
     [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]])

CarteBoucle = array([[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                      2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
                     [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
                      0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1],
                     [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                      2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]])

Fantôme1 = array(
    [[0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0]])
Fantôme2 = array(
    [[0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0]])
Fantôme3 = array(
    [[0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0]])
Fantôme4 = array(
    [[0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]])

Map_diag_fonda = array([[2 for k in range(0, 300)],
                        [0 for k in range(0, 300)],
                        [2 for k in range(0, 300)]])
```