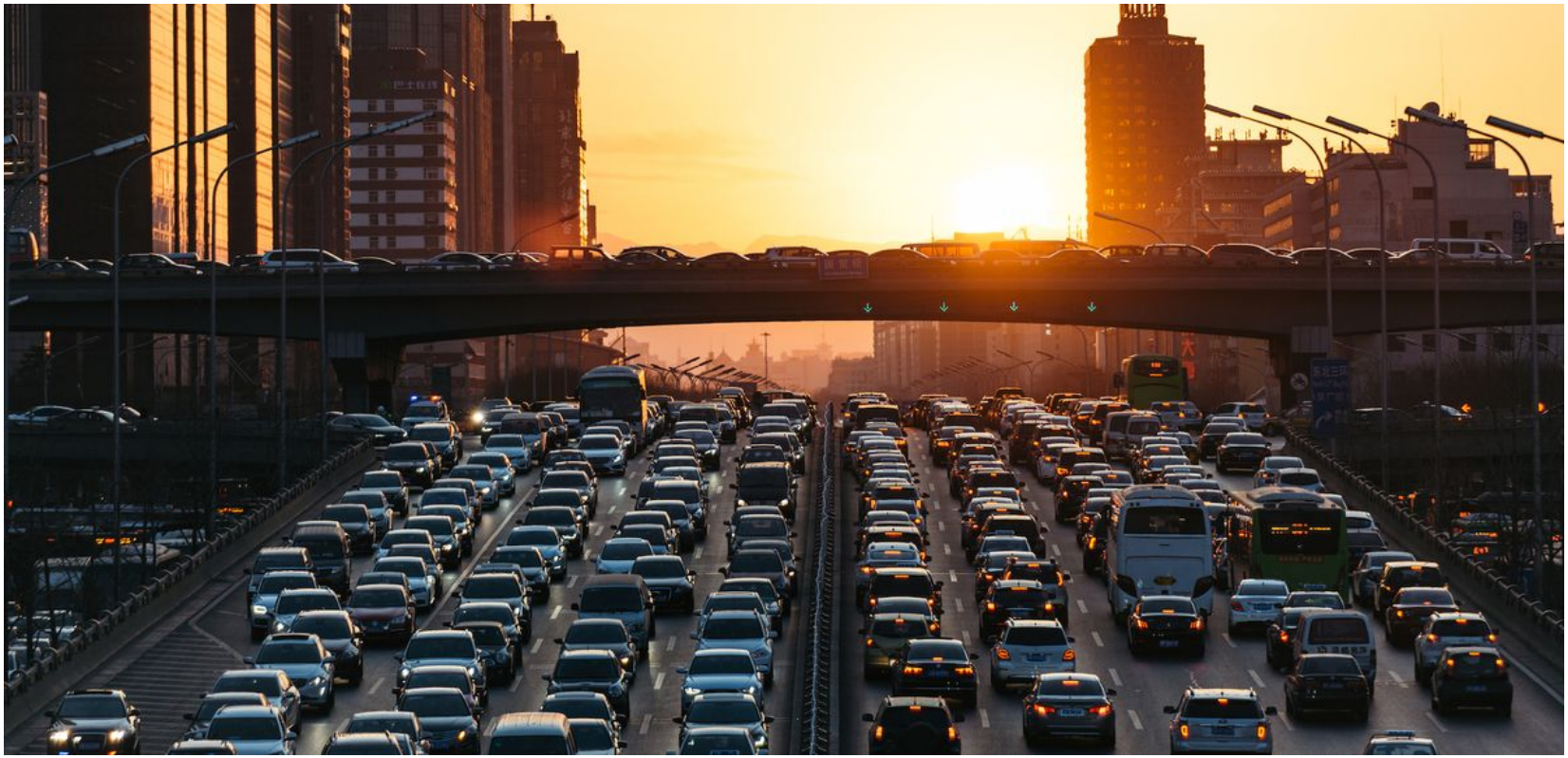
# Étude de l’impact des différents profils comportementaux des automobilistes sur la fluidité du trafic routier.



Alex Tual--Hamon - MPI - 2022/2023

numéro de candidat : 31941

# Plan

- Enjeux
- Problématique
- Objectifs
- Réalisation de l’application : implémentation et règles d’évolution
- Expériences et Conclusions

# 21e siècle et Urbanisation

Le monde s'urbanise :

- 1950 : 30% d'urbains → 2022 : 60 % d'urbains

De façon très concentrée :

- 1975 : 4 mégapoles → 2022 : 54 mégapoles

villes = pôle d'activité majeur → l'accessibilité y est essentielle

# Les embouteillages ont un coût

- Coût économique
- Coût en temps
- Coût écologique
- Coût sanitaire

# Problématique

Sur une route idéale (parfaitement droite, de largeur uniforme et sans obstacles ni perturbations extérieures),

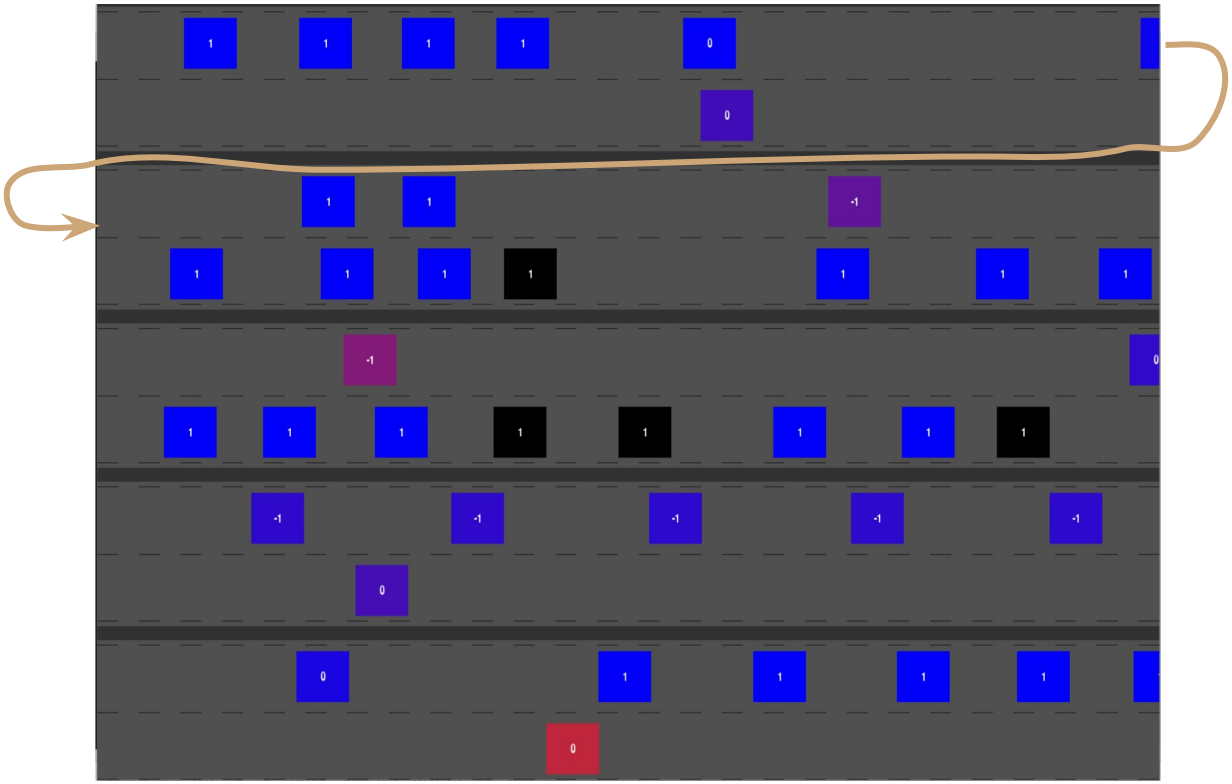
comment la fluidité du trafic routier est-elle impactée par les comportements des conducteurs ?

Que peut-on en déduire pour favoriser la fluidité du trafic ?

# Objectifs

Proposer puis tester des **solutions** afin de **prévenir** les embouteillages, grâce à une simulation

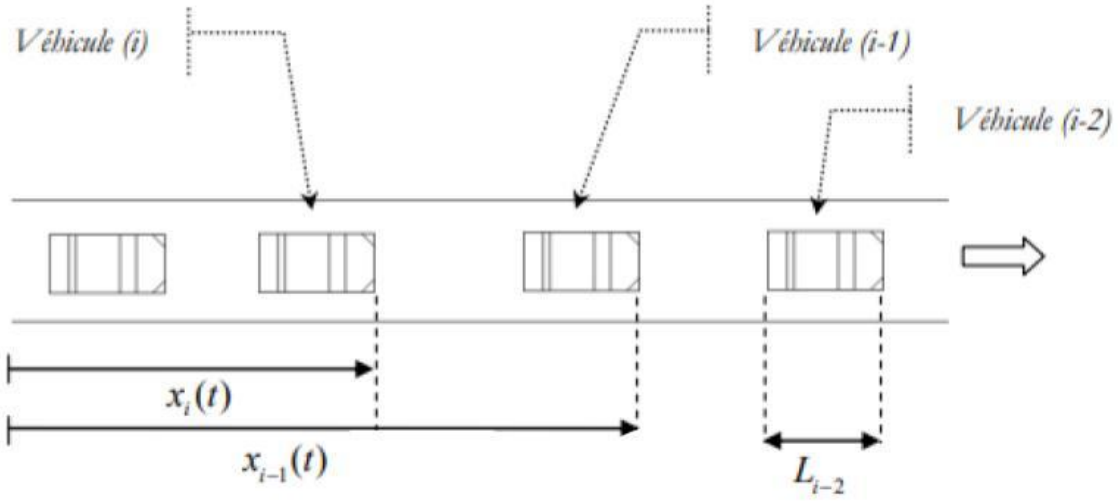
## Application : Simulation d'un trafic routier



Cahier des charges

- se concentre uniquement sur les causes internes
- affichage pour observer
- simule le comportement humain de façon réaliste
- paramétrable : on peut y entrer différentes situations
- retourne des données sur l'expérience effectuée

# Outils externes




$$a_i = a_{max}\left(1 - \left(\frac{v_i}{v_{souhaitée}}\right)^4 - \left(\frac{f(v_i,\Delta v_i)}{distance_{i->i+1}}\right)^2\right)$$

$$f(v_i,\Delta v_i) = distance_{min} + distance_{sécurité} + \frac{v_i \cdot \Delta v_i}{2\sqrt{a_{max} \cdot d_{confort}}}$$

# Arborescence des classes

## Objet **Route** : class road

- nombre de voies (int)
- vitesse maximale autorisée (m/s)
- taille (m)
- liste de liste chaînée “road.voies”

`[ voie1, voie2, …]`

## Objet **Cellule** : class cell

- position, vitesse, accélération
- véhicule associé (car)
- conducteur associé (driver)
- route de circulation (road)
- numéro de voie (int)
- données pour les états décisionnels
  - coefficients réels tirés au dans un intervalle propre au profil

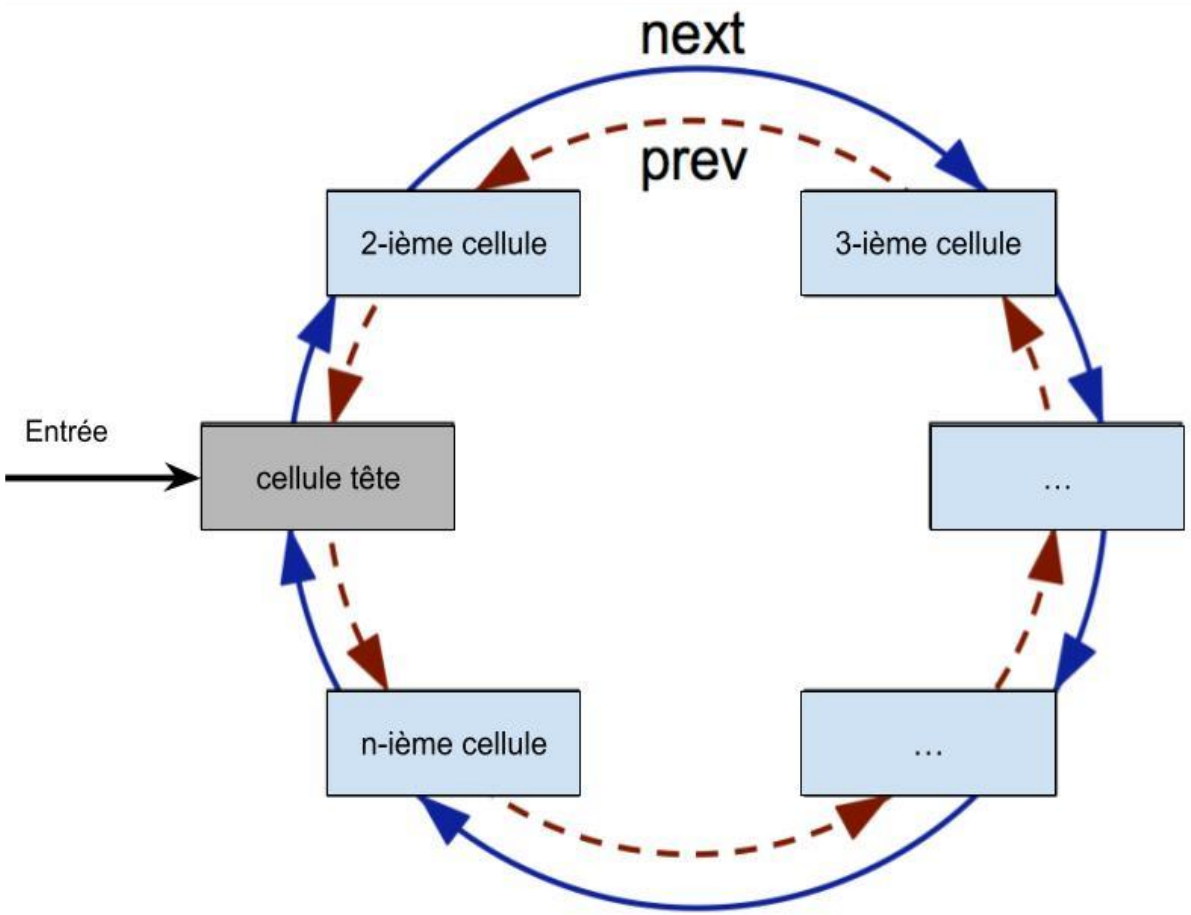
## Objet **Voiture** : class car

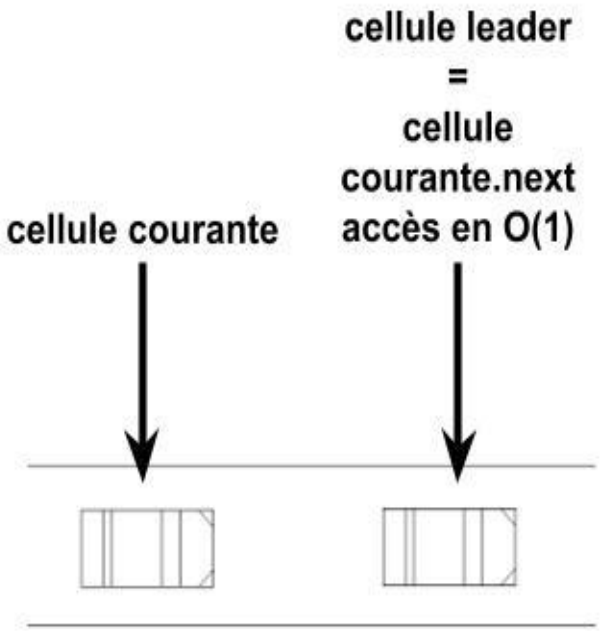
- accélération maximale ($m/s^2$)
- freinage maximal ($m/s^2$)
- longueur du véhicule (m)
- poid lourd ou non (bool)

## Objet **Conducteur** : class driver

- temps de réaction (s)
- distance de prevoyance (m)
- coefficients sur les paramètres
- paramètres des états de décisions
  - coefficients réels tirés au sort

# Implémentation en liste doublement chaînée



# Première implémentation de l'IDM

à chaque tour de boucle, on parcours l'ensemble des véhicule :

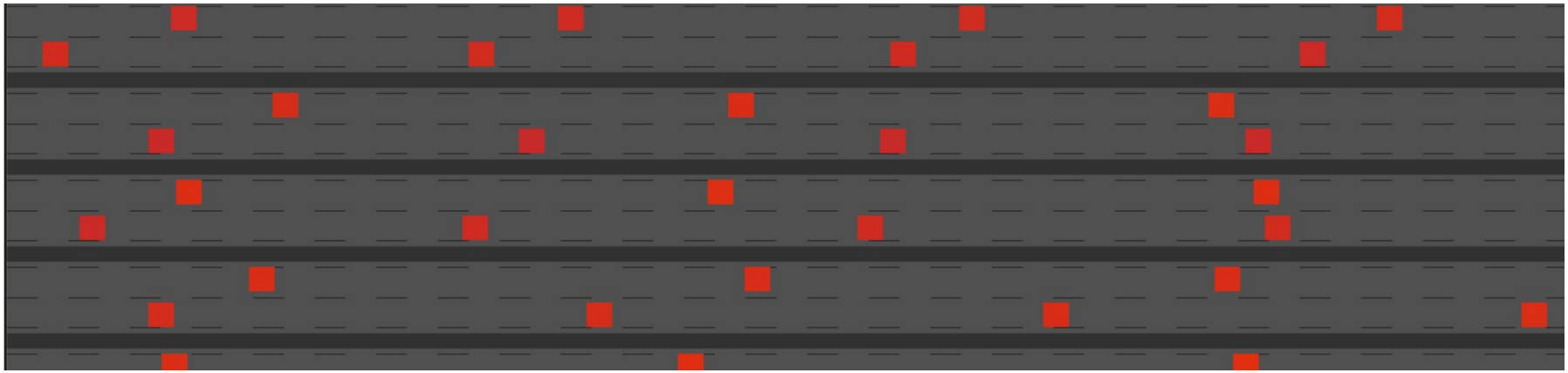
- mise à jour de la vitesse grâce à l'IDM

$$v_i(t + dt) = v_i(t) + a_i(t + dt) \cdot dt$$

- mise à jour de la position

$$x_i(t + dt) = x_i(t) + v_i(t + dt) \cdot dt$$

Où $dt$ est le temps d'un tour de boucle



# Modélisation du comportement humain : états décisionnels

Non circonstanciels : → apparaissent sans conditions sur le trafic

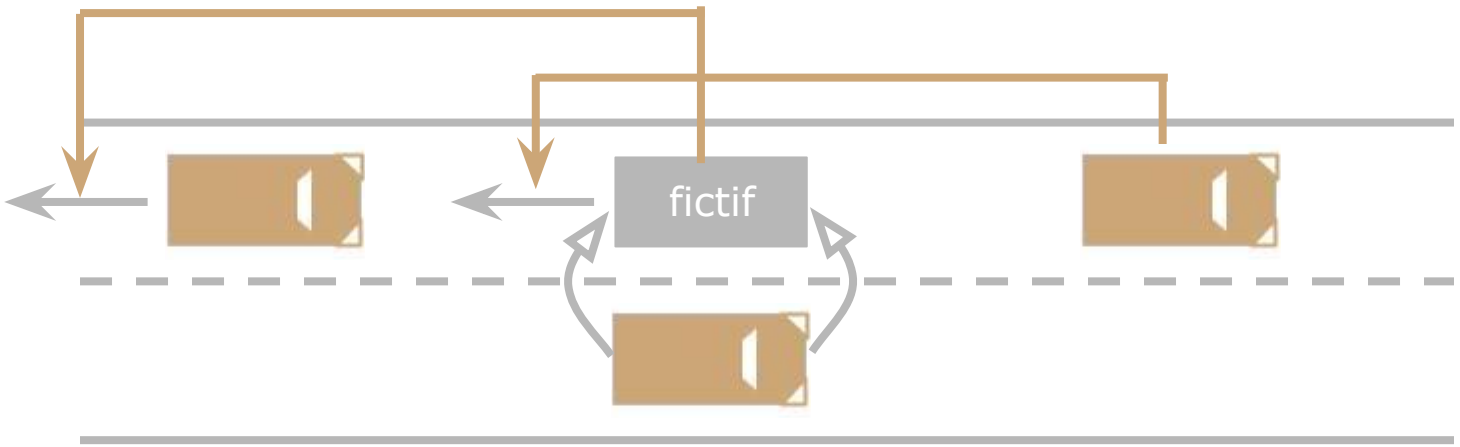
- accélération
- ralentissement
- neutre

Circonstanciels : → apparaissent sous certaines conditions

- Stop and go en régime lent
- changement de voie
- changement de voie en régime lent
- obstruction

# loi du mouvement : IDM modifié

- Pourquoi le modifier → comportement à hautes vitesses
  → changement de voie

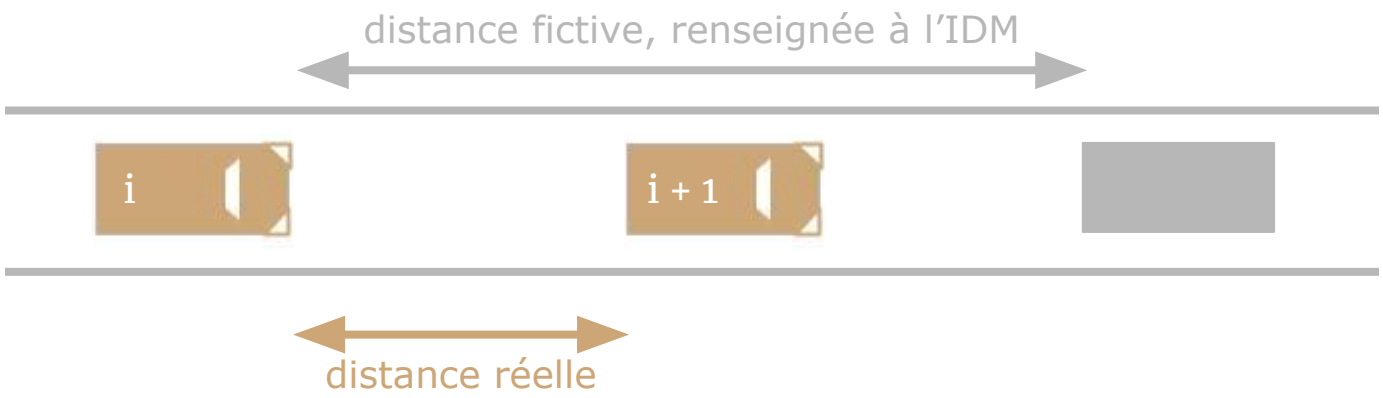



- comment
  → rendre l'IDM moins prudent à vitesses proches

Contraste de vitesse

- vitesses proches → 1
- vitesses éloignées → 0

# loi du mouvement : IDM modifié



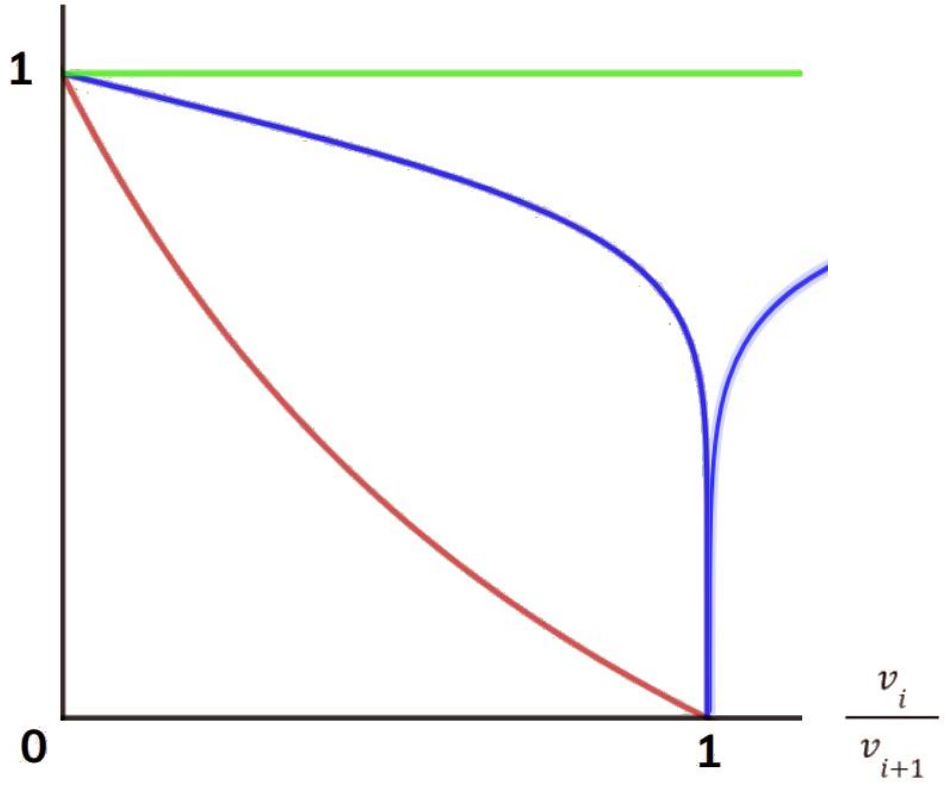

$$distance_{fictive} = \frac{distance_{réelle}}{C_v}$$

$$\frac{dv_i}{dt} = a\left(1 - \left(\frac{v_i}{v_0}\right)^4 - \left(\frac{d^*}{d_{i \to i+1} \div C_v}\right)^2\right)$$

# loi du mouvement : IDM modifié

Contraste :

$$C_v = \frac{|v_i - v_{i+1}|}{v_i + v_{i+1}}$$



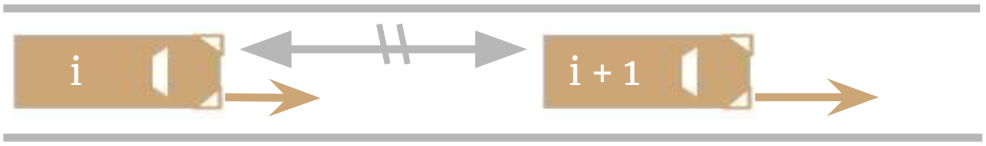

Pseudo contraste :

$$si \quad v_i \neq 0, \quad C_v = max(C_{min}, \sqrt[8]{\frac{|v_i - v_{i+1}|}{v_i + v_{i+1}}})$$

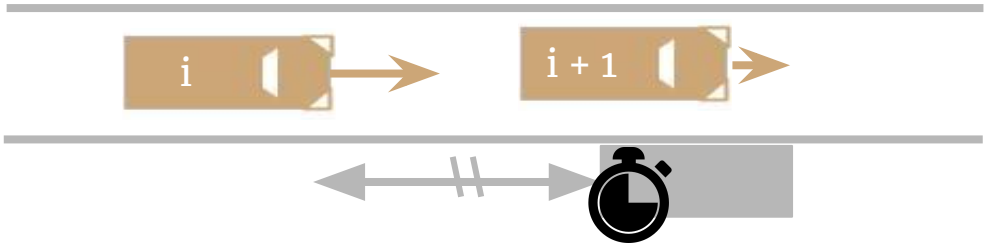
$$sinon, \quad C_v = 1$$

# Comportements humains : Temps de réaction

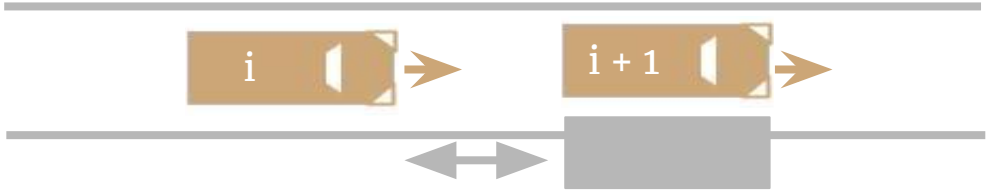
- Temps de réaction :




Phase stationnaire, peu de variations des vitesses




Variation du leader, i ne réagit pas encore et perçoit l'ancienne distance

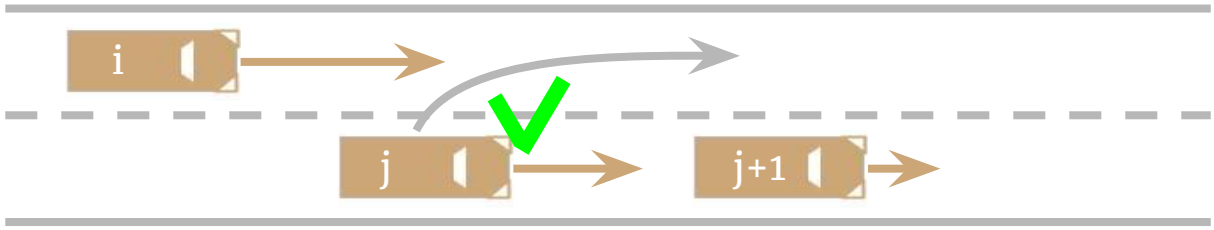



temps de réaction écoulé, i réagit, accède à la distance réelle

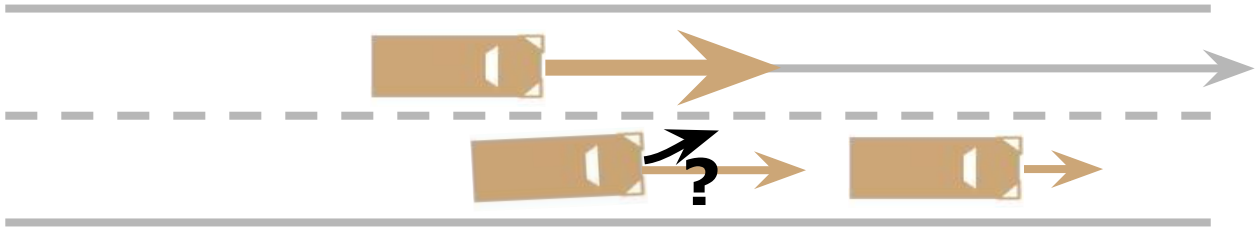
→ comportement adouci par un contraste d'accélération et un coefficient de respect des distances de sécurité

# Comportements humains : États décisionnels

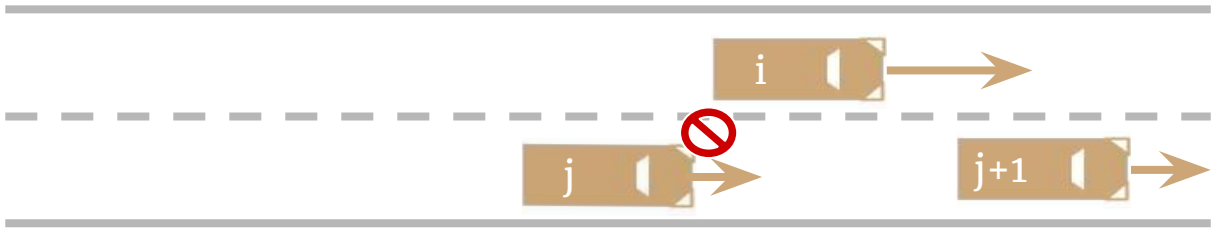
- État d'obstruction
- État de changement vers la voie gauche




j est ralentie par (j+1), elle entre dans l'état circonstanciel changement_voie_gauche



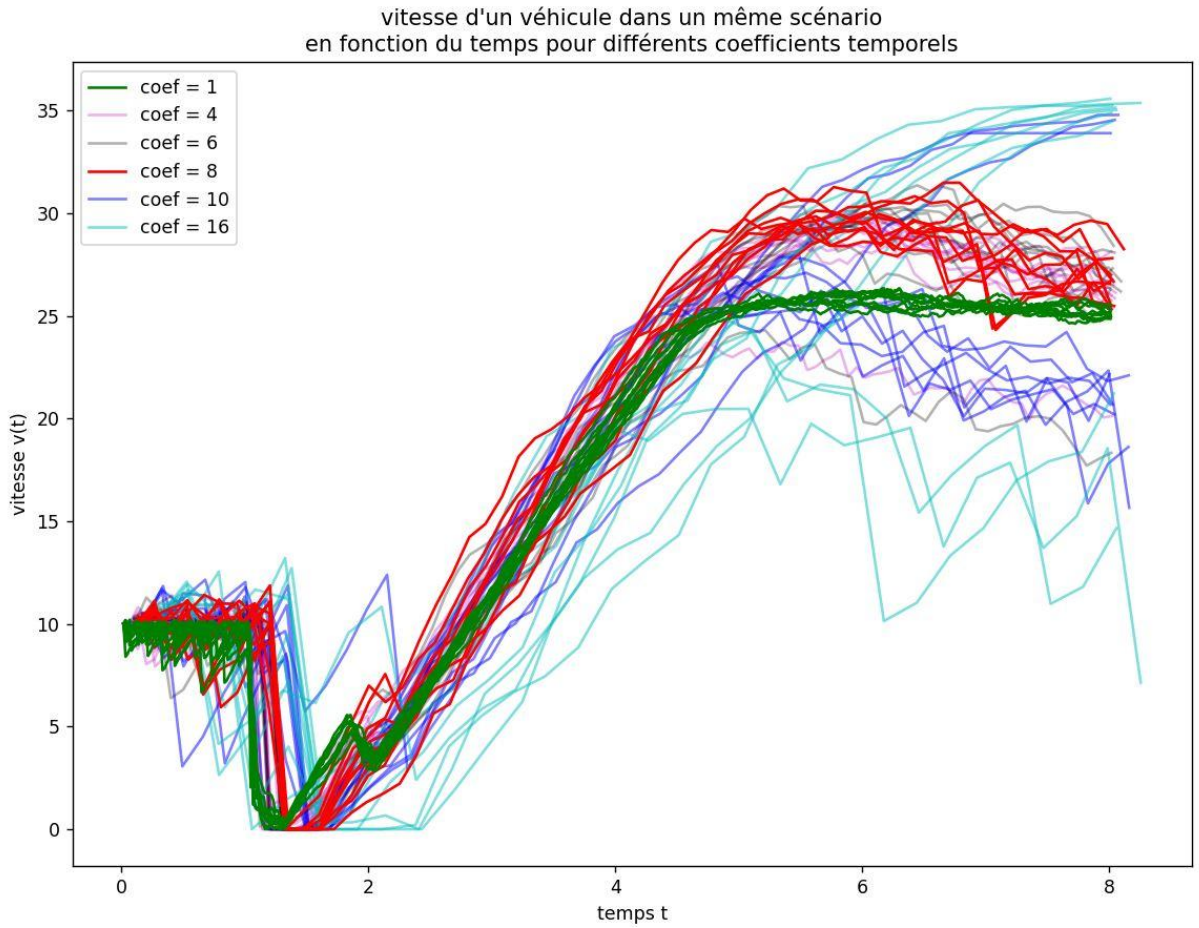

i détecte la volonté de changement de j → i accélère pour ne pas être obstrué par j




i est passée, j ne peux plus doubler car les conditions ne sont plus vérifiées

# Résolution de la modélisation

- Sensibilité à l'accélération temporelle et perte de résolution




```
C:\WINDOWS\SYSTEM32\cmd.exe
10
pygame 2.0.2 (SDL 2.0.16, Python 3.8.3)
Hello from the pygame community. https://

coef temporel = 10

10

coef temporel = 10

10

coef temporel = 10

ACCIDENT : test cell1


------------------
(program exited with code: 0)

Appuyez sur une touche pour continuer...
```

$$dt \leftarrow dt \cdot coefficient\ temporel$$

# Protocole expérimental

Objectif : prévenir les embouteillages → on part d'une vitesse moyenne élevée et d'une forte densité

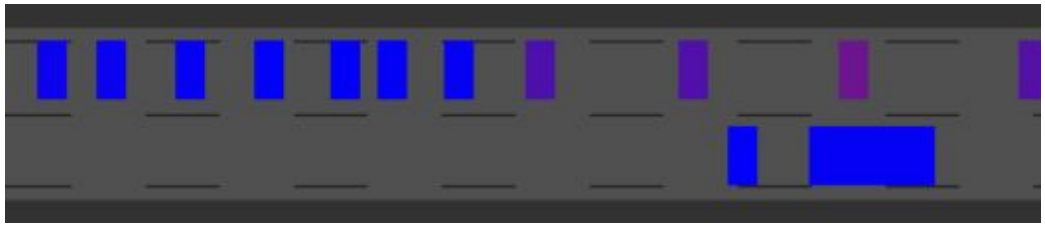
Indicateurs : vitesse moyenne et nombre de véhicule par tranches de vitesse

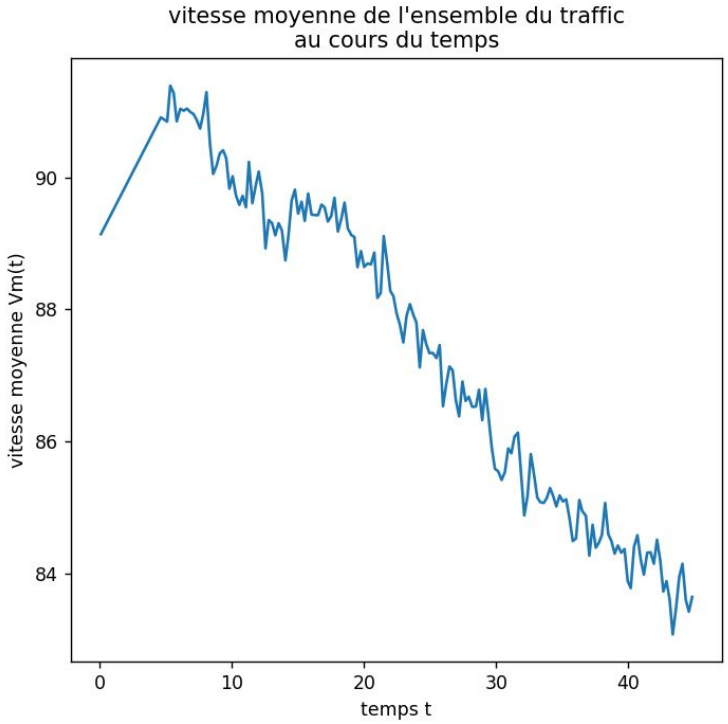
- déterminer les conditions d'expérimentation optimales
- se concentrer sur des causes potentielles
- implémenter des solutions à travers les paramètres
- comparer les résultats

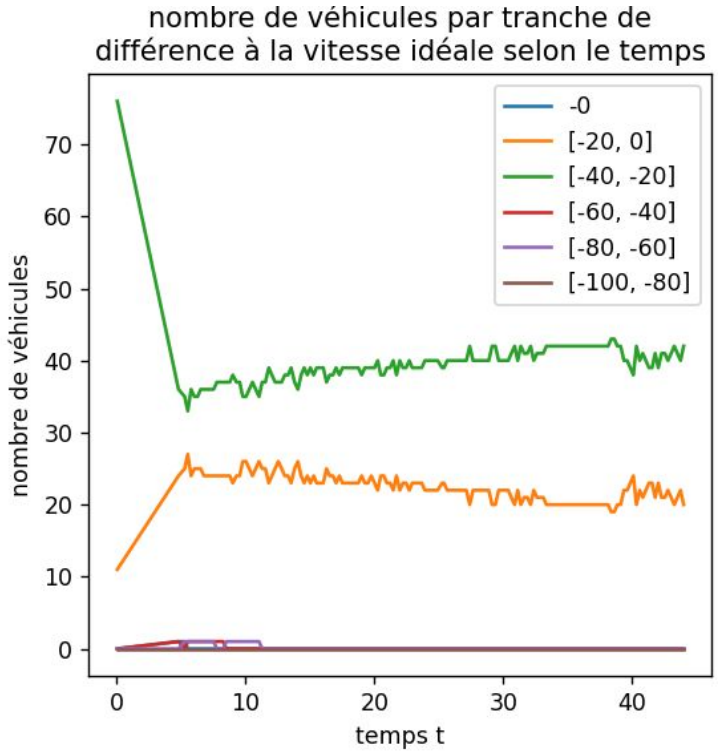
Yuki Sugiyama
Université de Nagoya
article : Traffic jams without bottlenecks—experimental evidence for the physical mechanism of the formation of a jam

# Impact des poids lourds

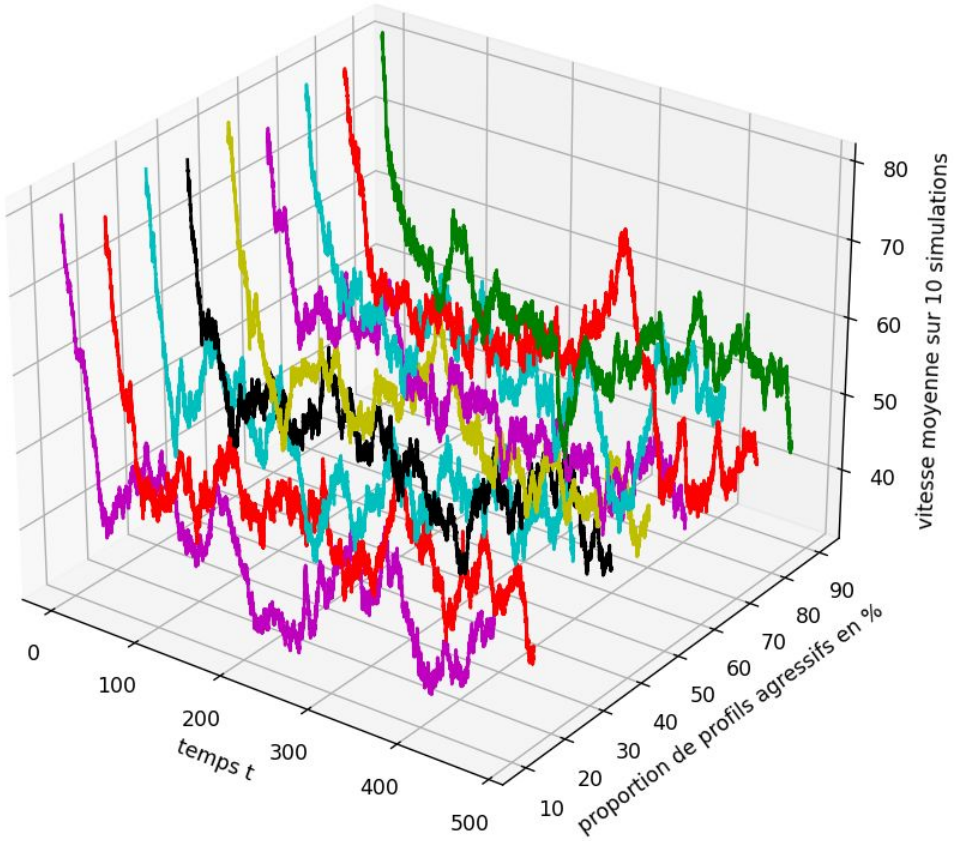
- route de 3 km
- 16% de camion



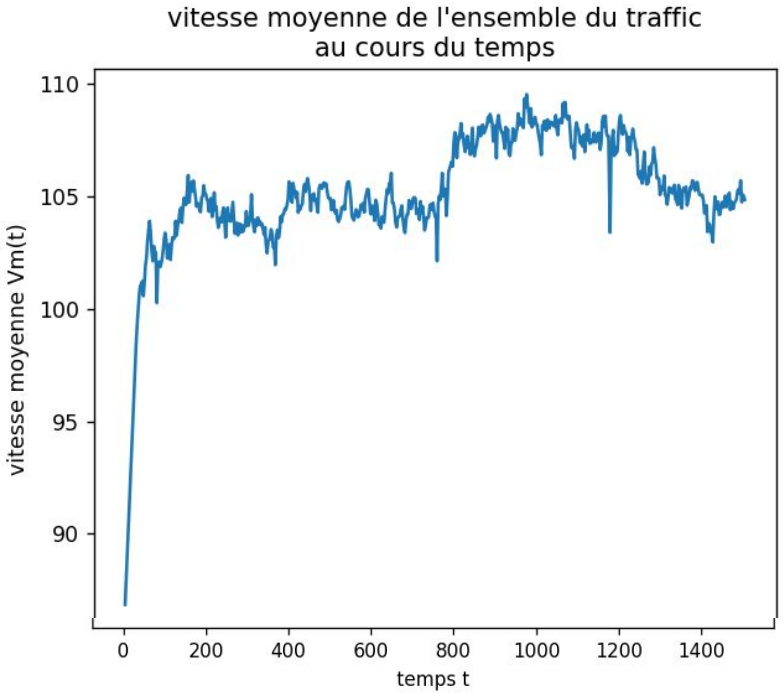


# Suppression ou limitation des poids lourds

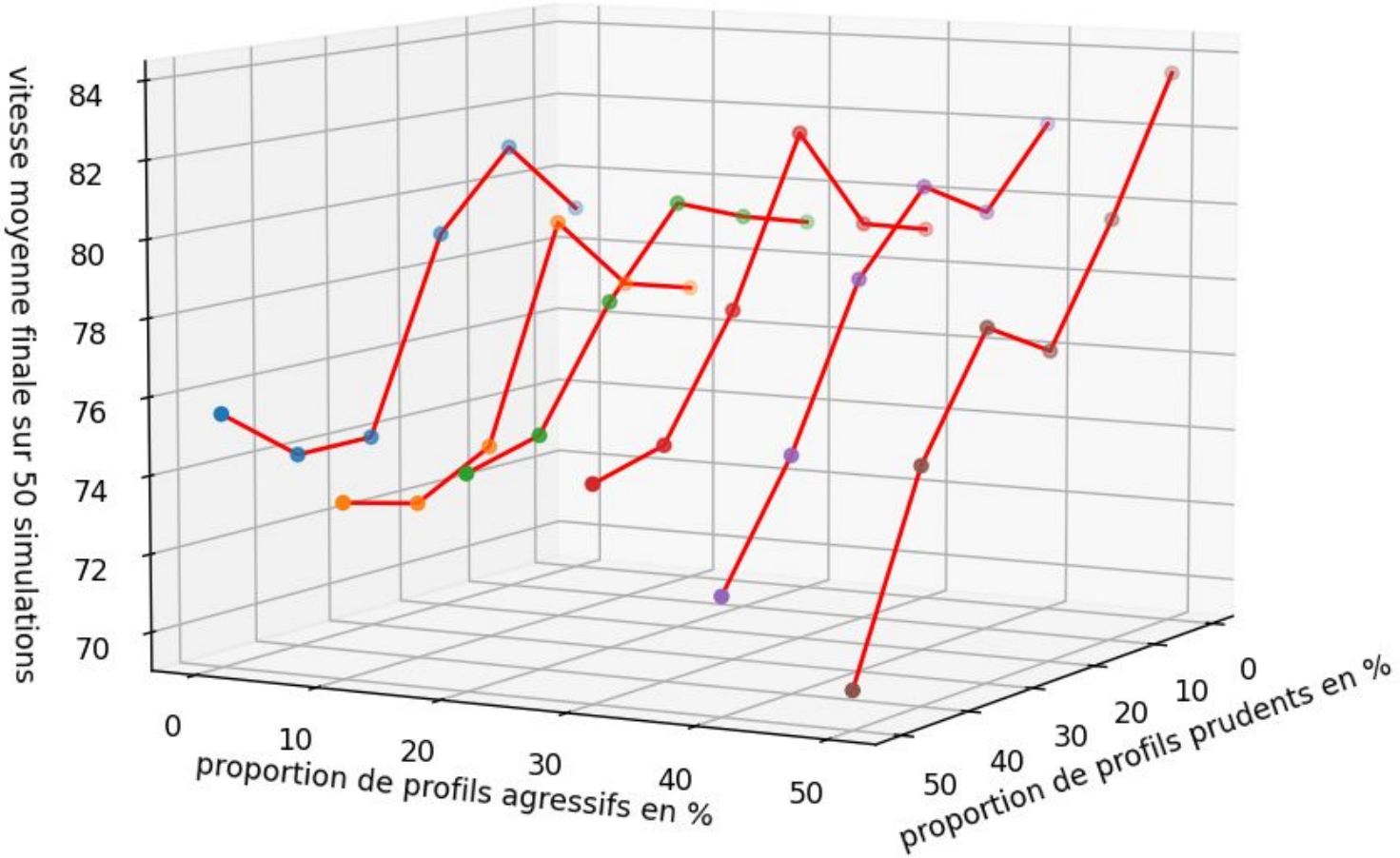



moyenne sur 10 simulations de la vitesse moyenne selon le temps, selon la proportion de conducteurs agressifs avec camions




Vitesse moyenne sur 50 simulations
Sans camions

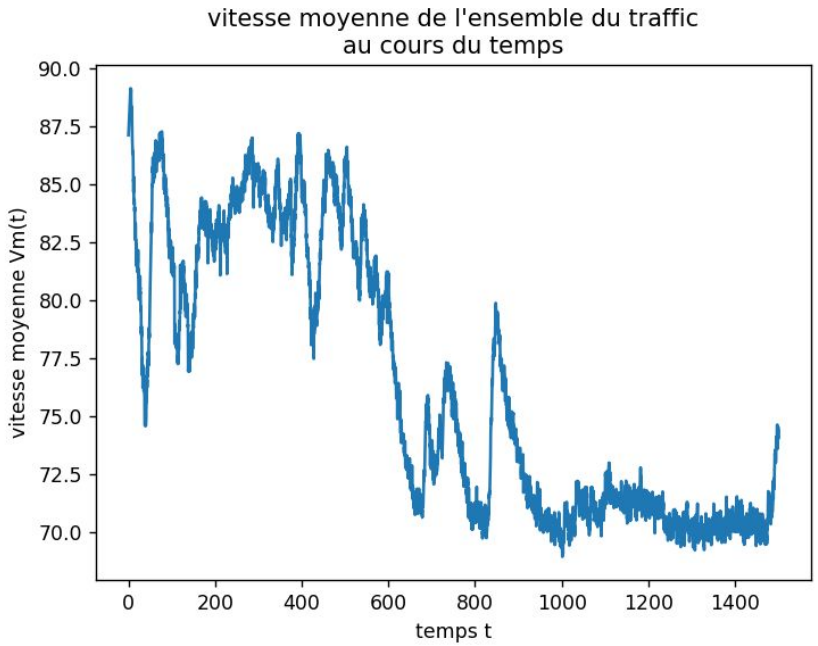
# Profils comportementaux : proportion optimale

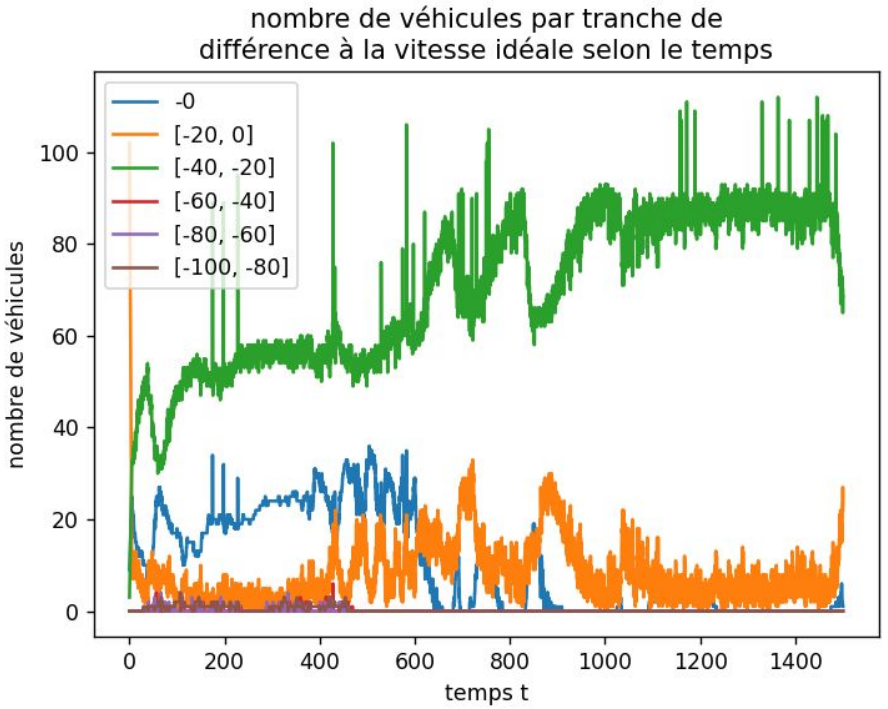



Vitesse moyenne dans l'état stationnaire final après 1400 secondes de simulation, moyenne de 50 simulations, proportion de moyen en complémentaire

# Limitations variables et régulateur de vitesse

- route de 4 km
- 120 véhicules variés
- limitation : 130 km/h
- vitesse idéale : 110 km/h



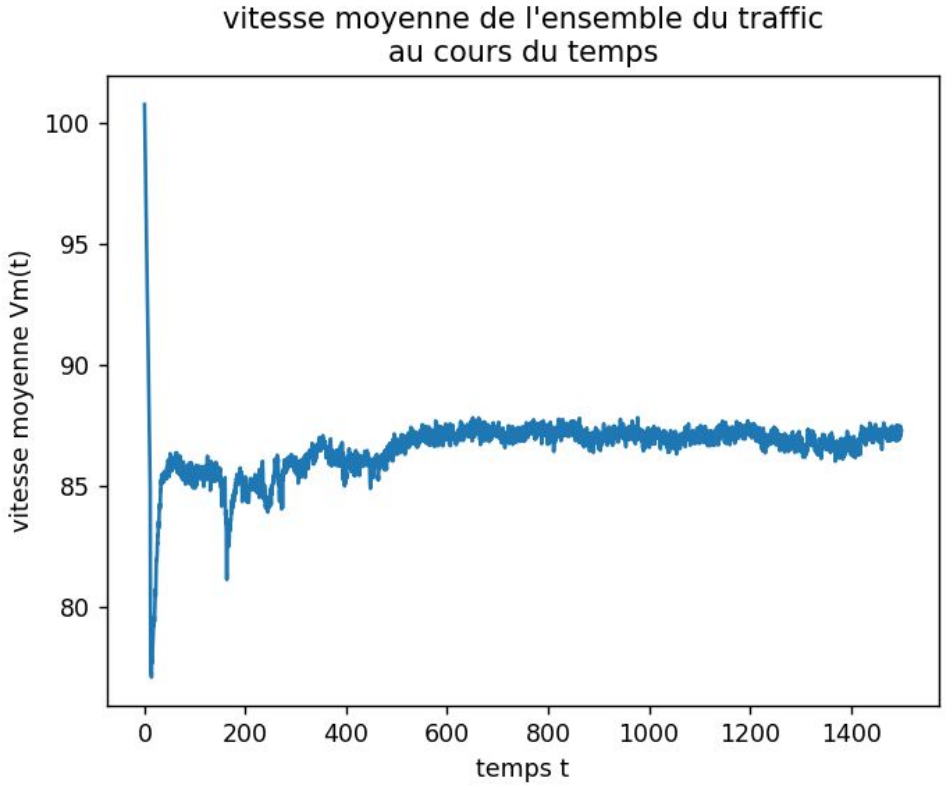




moyenne sur 50 simulations, on retrouve le même scénario
(⅓ de chaque profil)

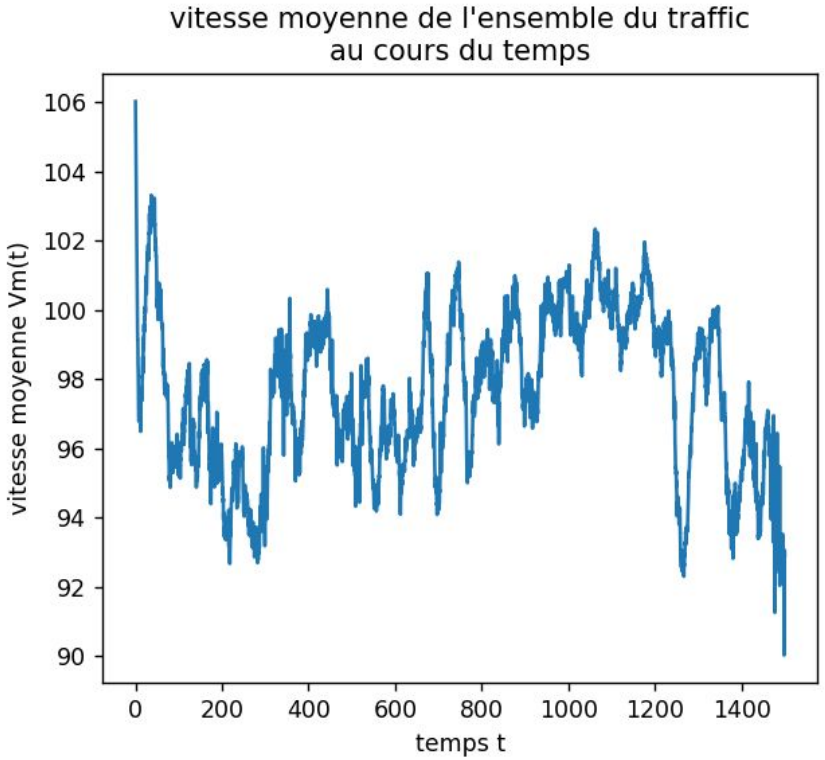
# Limitations variables et régulateur de vitesse

implémentation :

- vitesse stabilisée à l'entrée en régime neutre et en régime précédent la réaction
- abaissement de la vitesse limite à la vitesse idéale pour la densité donnée

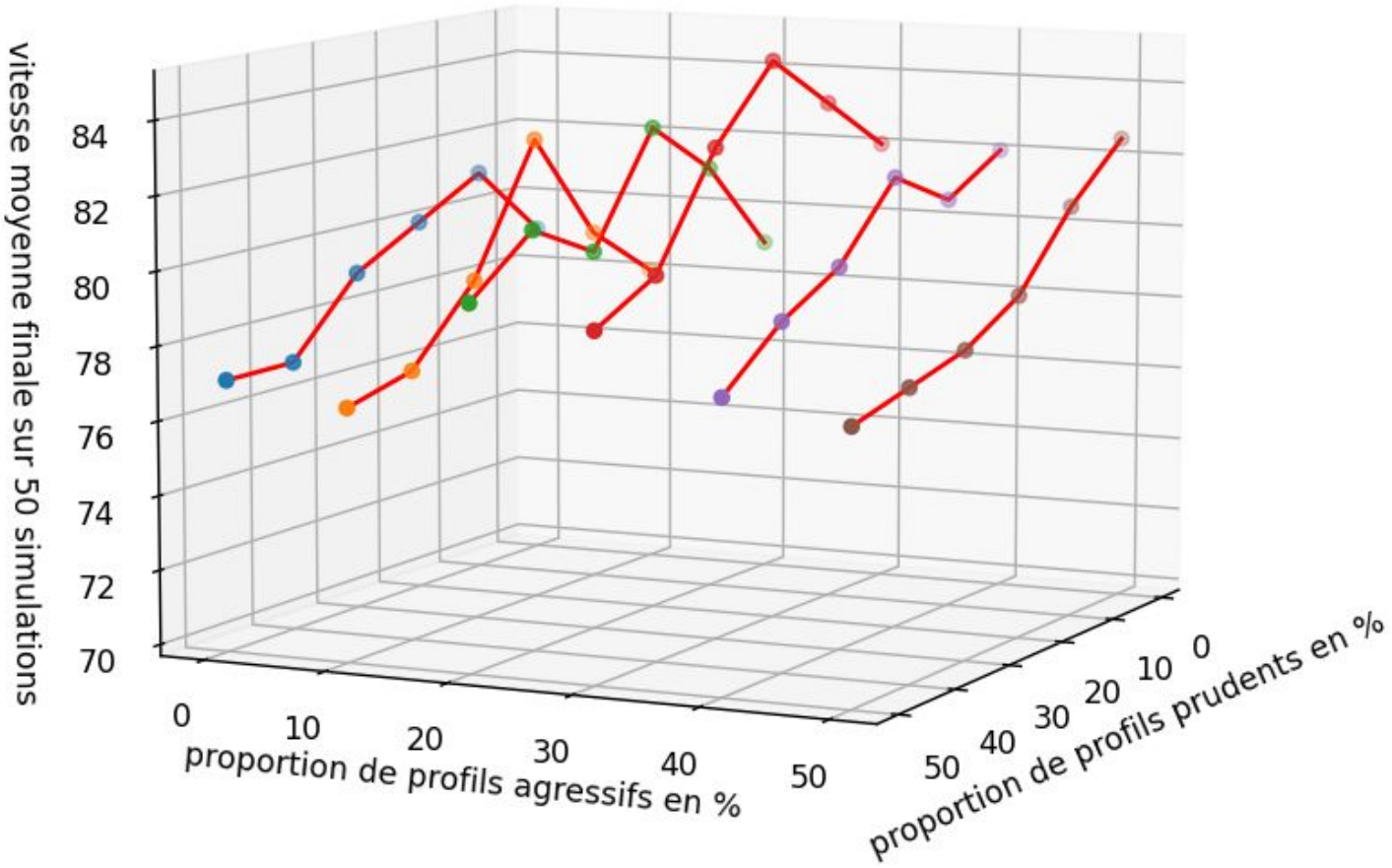



limitation variable à 90 km/h




limitation variable à 110 km/h
→ vitesse idéale

# Limitations variables et régulateur de vitesse




limitation variable à 90 km/h, scénarios plus regroupés, moins chaotiques et même maximum

# Solutions

- échelle individuelle : imposer/exclure des comportements
  - limiter son accélération, son freinage
  - rendre les conditions de dépassement plus prudentes
  - respecter les limitations provisoires imposées
  - changer de voie avec l'ambition d'étaler la densité
  - activer son régulateur de vitesse
- échelle collective :
  - changer la vitesse max. autorisée selon la densité
  - connecter les voitures entre elles → conduite automatique, répartition spatiale
  - limiter, interdire les camions

→ **limiter les hétérogénéités**

Les conditions imposées pour prévenir les embouteillages ne sont pas forcément

les meilleures pour les désamorcer

Sans oublier les solutions externes

- utiliser les transports en commun/covoiturage
- ajouter des voies (mais paradoxe de Braess)
- décaler les horaires de travail
- aménagements cyclables

# Annexe

Sources :

- papier de l'Asfa (Association des sociétés françaises d'autoroutes) : autoroutes.fr
- Le Figaro : lefigaro.fr
- BfmTV : bfmtv.com
- Wikipedia : wikipedia.org

# Code source

```
def IDM(self,cell,front_cell,distance, DV=None):
            #déclarations par défaut
            if DV == None:      DV = cell.content.v - front_cell.content.v
            #définition contraste de vitesse
            if cell.content.v != 0:    CV = max(0.00000000001,(DV)/(cell.content.v + front_cell.content.v))
            else:  CV = 1
            if distance < cell.content.v*2:  respect_distance_secu = cell.content.v*2/distance
            else:  respect_distance_secu = 1
            CV = CV**(1/8)*respect_distance_secu #pseudo contraste de vitesse, pour exacerber les petits contrastes et
ainsi tendre rapidement vers l'IDM

            Did = cell.content.driver.coeff_distance_secu * (2 + max(0,
cell.content.v*1.5+(cell.content.v*(DV)/(2*sqrt(cell.content.driver.coeff_acceleration*cell.content.car.acceleration_max*
cell.content.car.freinage_max*cell.content.driver.coeff_deceleration)))))
            if Did < cell.content.car.longueur * 1000 : #on majore la distance idéale par 5000 (arbitraire, juste grand
devant la taille du problème)
                  interact = (  Did  /  (distance/CV)  )**2
                  a = cell.content.driver.coeff_acceleration * cell.content.car.acceleration_max   *   (  1 - (
cell.content.v  /  (cell.content.driver.vitesse_voulue) )**4  -   interact)
            else :
                  a = -(cell.content.driver.coeff_deceleration * cell.content.car.freinage_max) #on minore l'accell par
le freinage max
            return a
```

```
class DCList: #Double Chained List
      def __init__(self,content, previous = None, next = None): # content : some_type, previous :
DCList, next : DCList
            self.content = content
            self.previous = previous
            self.next = next

import pygame
from pygame.locals import *
from classes import Road,Driver,Car,Cell
# ~ HEIGHT,WIDTH = 830,1530
# ~ WINDOW = pygame.display.set_mode((WIDTH, HEIGHT)) #Setting WINDOW
# ~ pygame.display.set_caption('Traffic SIM') #Window Name
#_____________________________________________________________________________
def displayCalculs(road:Road, dim:tuple):
      HEIGHT, WIDTH = dim
      taille = road.taille

      if taille < 1000 :
            taillePort = round( taille / 5 )
      if 1000 <= taille <= 5000 and len(road.voies)<=3  :
            taillePort = round( taille / 10 )
      if 1000 <= taille <= 5000 and len(road.voies)>3  :
            taillePort = round( taille / 5 )
      if taille > 5000 and len(road.voies)<=2  :
            taillePort = round( taille / 10 )
```

```
        if taille > 5000 and len(road.voies)>2 :
                taillePort = round( taille / 5 )
        nbPortions = round( taille / taillePort )
        gap = 15
        if len(road.voies) == 1:
                gap = 50
        roadTness = round( (HEIGHT-((nbPortions-1)*gap)) / nbPortions ) # largeur route (ie tte les voies)
        tness = round ( roadTness/len(road.voies) )


        rep : tuple = (taillePort, nbPortions, gap, roadTness, tness)
        return rep
#_____________________________________________________________________________ drawRoad

def drawRoad(road, WINDOW, dim : tuple):
        HEIGHT, WIDTH = dim
        ROADCOLOR = (80, 80, 80) #gris
        WHITE = (35, 35, 35) #définition du blanc (gris sombre)
        taille = road.taille

        taillePort, nbPortions, gap, roadTness, tness = displayCalculs(road, dim)

        yDraw = 5
        for j in range(nbPortions):
                # DESSINE PAR PORTIONS
                pygame.draw.rect(WINDOW, ROADCOLOR, (0,yDraw,WIDTH,roadTness)) #Drawing the road, thickness 0 = filled, 0 pos en x
#_____________________________________________________________________________
                #dessin des lignes de sécurité blanches
                x=0
                sideLen = 30 #longueur dans le sens de la route des bandes
                sideWid = 1 #largeur des bandes en pixel
                while x < WIDTH: #boucle qui fait avancer la zone de dessin en x (horizontalement)
                        pygame.draw.rect(WINDOW, WHITE, (x,yDraw+5, sideLen,sideWid))#en haut
                        pygame.draw.rect(WINDOW, WHITE, (x,yDraw + roadTness - 5 - sideWid, sideLen,sideWid))#en bas
                        i=0
```

```
while i<(len(road.voies)-1): #boucle qui dessine chaque étage de bande blanche en un même x
                            y = yDraw+(tness*(i+1))-(sideWid//2)
                            pygame.draw.rect(WINDOW, WHITE,(x, y, sideLen, sideWid))#au milieu, entre les voies
                            i += 1
                    x+=2*sideLen
            #met à jour yDraw, on dessine la portion suivante au prochain tour de boucle avec cette nouvelle position
            yDraw += roadTness + gap



#_____________________________________________________________________________
#_____________________________________________________________________ drawCell

def drawCell(road, WINDOW, dim : tuple): #dessine chaque véhicule
        #CONSTANTES
        HEIGHT,WIDTH = dim #taille de la fenêtre
        taille = road.taille

        taillePort, nbPortions, gap, roadTness, tness = displayCalculs(road, dim)

        portion = taillePort #en mètre
        coef = round(WIDTH/portion)
        for i in range(len(road.voies)):
                cell = road.voies[i]
                head = road.voies[i]

                while cell.next != head and cell.next != None:
                        x = cell.content.x
                        indPort = (x // portion)
                        yDraw = ((tness//2) - round((tness*0.7)/2)) +      +5 + indPort*(gap+roadTness)      + tness*i
                        # -round(..)/2 car on va en haut du véhicule de largeur round(..), +20 car décalage initial, 2 terme pr la
portion, init à 20, 3e pr la voir dans la portion1
                        xPort = x - portion*indPort
                        xDraw = round(xPort * coef)
                        pygame.draw.rect(WINDOW,cell.content.car.color,(xDraw, yDraw, coef*cell.content.car.longueur,
round(tness*0.7))) # le dernier nbre donne la largeur du véhicule, 80% de la largeur de la voie
```

```
font = pygame.font.SysFont(None, cell.content.car.longueur*5)
                    img = font.render(str(cell.content.name), True, (255, 255, 255))
                    rect = img.get_rect()
                    rect.center = (xDraw+(coef*(cell.content.car.longueur))//2, yDraw+(round(tness*0.7))//2)
                    WINDOW.blit(img, rect)

                    cell = cell.next

             if cell.content != None:
                    x = cell.content.x
                    indPort = (x // portion)
                    yDraw = ((tness//2) - round((tness*0.7)/2)) +      +5 + indPort*(gap+roadTness)      + tness*i
                    # -round(..)/2 car on va en haut du véhicule de largeur round(..), +20 car décalage initial, 2 terme pr la
portion, init à 20, 3e pr la voir dans la portion1
                    xPort = x - portion*indPort
                    xDraw = round(xPort * coef)
                    pygame.draw.rect(WINDOW,cell.content.car.color,(xDraw, yDraw, coef*cell.content.car.longueur,
round(tness*0.7))) # le dernier nbre donne la largeur du véhicule, 80% de la largeur de la voie

                    font = pygame.font.SysFont(None, cell.content.car.longueur*5)
                    img = font.render(str(cell.content.name), True, (255, 255, 255))
                    rect = img.get_rect()
                    rect.center = (xDraw+(coef*(cell.content.car.longueur))//2, yDraw+(round(tness*0.7))//2)
                    WINDOW.blit(img, rect)

                    if cell.content == None: s += "\tempty\n"
                    else:
                          while cell.next != None:
                                s += str(cell.content) + "\n"
                                cell = cell.next
                          voie += 1
             return s

      def accidentManager(self,cell):
             print("ACCIDENT : " + cell.content.name)
             self.list_accidents.append(str(cell.content.name))
             front = cell.next
             cell.previous.next = front
             front.previous = cell.previous

             # ~ run = True
             # ~ while run:
                    # ~ if win32api.GetKeyState(0x1B) < 0:      run = False #échap met fin a la boucle
while sans fermer la fenêtre
                    # ~ sys.exit()
```

```
    def distance_inter_cell(self,cell,front_cell):
        if front_cell.content != None and cell.content != None :
            if front_cell.content.x + front_cell.content.car.longueur < cell.content.x:
distance = self.taille - cell.content.x - cell.content.car.longueur + front_cell.content.x
            else: distance = front_cell.content.x-cell.content.x-cell.content.car.longueur
            return max(0.2,distance)
        return 500 #équivalent de l'infini devant la taille du problème, si ya pas de
front_cell
#_____________________________________________
    def change(self,cell,side_back_cell,voie : int): #retourne la tête de liste chainée
        if voie == 1:   cell.content.driver.vitesse_voulue =
cell.content.driver.vitesse_moyenne*cell.content.car.limit
        ancienne_voie = cell.content.voie
        content = cell.content
        if cell.next == cell: #si la cellule était seule sur sa voie
            self.voies[ancienne_voie].content = None
            self.voies[ancienne_voie].next = None
            self.voies[ancienne_voie].previous = None
        else: #sinon si la cellule n'était pas seule sur sa voie
            cell.next.previous = cell.previous
            cell.previous.next = cell.next
            #si la cellule était la tête maintient de la tête dans l'ancinne voie
            if cell == self.voies[ancienne_voie]:
                if cell.previous.content.x < cell.content.x:
                    self.voies[ancienne_voie] = cell.previous
                else: self.voies[ancienne_voie] = cell.next
        post_cell = DCList(content)
        if side_back_cell.content == None: #si la cellule est seule sur la nouvelle voie
            post_cell.next = post_cell
            post_cell.previous = post_cell
            self.voies[voie] = post_cell
        else:
            post_cell.previous = side_back_cell
            post_cell.next = side_back_cell.next
            post_cell.next.previous = post_cell
            post_cell.previous.next = post_cell
        post_cell.content.voie = voie
        # ~ if self.voies[voie].previous.content.x < self.voies[voie].content.x:
```

```
self.voies[voie] = self.voies[voie].previous
            return self.voies[ancienne_voie]

      def changeManager(self,cell,side_front_cell): #retourne la tête de liste chainée/peloton
            distance_limite = 4
            if cell.content.car.camion == True :
                  distance_limite = 8
            decel_limite_changement = cell.content.driver.decel_limite_changement*coefT
            #vérification des préconditions sur cell.content.change (voie demandée existante)
            if cell.content.change < 0 or cell.content.change > len(self.voies)-1 or cell.content.change ==
cell.content.voie:
                  cell.content.change = -1
                  return self.voies[cell.content.voie] #on retourne la tete de liste inchangée
            #fin vérifications
            #début du changement de voie
            elif cell.content.voie < cell.content.change:  step = cell.content.voie + 1
            else: step = cell.content.voie - 1 #assignation de l'indice de la prochaine voie à atteindre à la variable
step
            #on trouve la cellule leader sur la voie step
            side_back_cell = side_front_cell.previous
            if side_front_cell.content == None:     return self.change(cell,side_front_cell,step)

            #calcul des conditions de changement de voie

            #calcul condition voiture de devant
            #calcul distance
            distance = self.distance_inter_cell(cell,side_front_cell)
            #calcul acceleraction
            if distance < distance_limite:   a_fictive_next = decel_limite_changement # équivalent à false
            else:  a_fictive_next = self.IDM(cell,side_front_cell,distance)*coefT

            #calcul condi tion voiture de derrière
            #calcul distance
            distance2 = self.distance_inter_cell(side_back_cell,cell)
            #calcul acceleraction
            if distance2 < distance_limite:  a_fictive_previous = decel_limite_changement
            else:  a_fictive_previous = self.IDM(side_back_cell,cell,distance2)*coefT

            #vérification des condition de déportation sur la voie step
            # ~ print(side_back_cell.content.name,distance2,side_front_cell.content.name,distance
            # ~ if cell.content.name == "test_cell50":
                  # ~ print("\n")
                  # ~ print(a_fictive_previous)
                  # ~ print(a_fictive_next)
```

```
            if a_fictive_next > decel_limite_changement and a_fictive_previous > decel_limite_changement: return
self.change(cell,side_back_cell,step)
            else:  return self.voies[cell.content.voie]
            # ~ return self.change(cell,side_back_cell,step)

      def IDM(self,cell,front_cell,distance, DV=None):
            distance += 0.2
            #déclarations par défaut
            if DV == None:      DV = cell.content.v - front_cell.content.v

            #définition contraste de vitesse
            if cell.content.v != 0:    CV = max(0.00000000001,(DV)/(cell.content.v + front_cell.content.v))
            else:  CV = 1 # qd la vitesse est nulle

      #cas de distance faible : on augmente le contraste, on le rapproche de 1, pr PLUS d'IDM
            if distance < cell.content.v*cell.content.driver.temps_distance_secu_CV/coefT:  respect_distance_secu =
cell.content.v*cell.content.driver.temps_distance_secu_CV/coefT/(distance)
            else:  respect_distance_secu = 1
            CV = CV**(1/(8))*respect_distance_secu #pseudo contraste de vitesse, pour exacerber les petits contrastes et
ainsi tendre rapidement vers l'IDM
      #cas de distance faible : on augmente le contraste, on le rapproche de 1, pr PLUS d'IDM

            Did = cell.content.driver.coeff_distance_secu * (2 + max(0,
cell.content.v*cell.content.driver.reaction/coefT+(cell.content.v*(DV)/(2*sqrt(cell.content.driver.coeff_acceleration*cel
l.content.car.acceleration_max*(coefT**2)*cell.content.car.freinage_max*cell.content.driver.coeff_deceleration)))))
            if Did < cell.content.car.longueur * 1000: #on majore la distance idéale par 500(arbitraire, juste grand
devant la taille du problème)
                  interact = (  Did  /  (distance/CV)  )**2 #on diminue l'impression de ditance --> plus raisonnable'
                  if cell.content.car.camion == True :a = cell.content.driver.coeff_acceleration *
cell.content.car.acceleration_max   *   (  1 - (  cell.content.v  /  (cell.content.car.limit*coefT) )**4  -   interact)
                  else : a = cell.content.driver.coeff_acceleration * cell.content.car.acceleration_max  * coefT *   (
1 - (  cell.content.v  /  (cell.content.driver.vitesse_voulue*coefT) )**4  -   interact)
            else :
                  a = min(front_cell.content.a-2*coefT, -(cell.content.driver.coeff_deceleration *
cell.content.car.freinage_max*coefT)) #on minore l'accell par le freinage max

            return a

      def find_leader(self,cell,voie):
            back_cell = self.voies[voie]
            head = self.voies[voie]
            if head.content == None: #si la voie step est vide
                  return head
            else: #sinon, on itère sur les cellules jusqu'a trouver celles entre lesquelles s'intercaler
                  if head.previous.content.x < head.content.x:   head = head.previous #maintient de la tête de liste
comme cellule d'abscisse minimale
                  if back_cell.content.x >= cell.content.x:
```

```
back_cell = back_cell.previous
                    while back_cell.next != head and back_cell.next.content.x < cell.content.x:
                          back_cell = back_cell.next
                    front_cell = back_cell.next
             return front_cell

      def gestionaire_evenements(self,dt,cell,front_cell,distance):
             DV = None
             if cell.content.TCDC[0] + cell.content.TCDC[1] < time.time(): #cooldown  est-ce le temps de changer d'état ?
si oui allons-y
                    #on aura besoin de ça :
                    side_front_cell = self.find_leader(cell,(cell.content.voie+1)%2)
                    dist_to_side_front = self.distance_inter_cell(cell,side_front_cell)
      #_____________________________________
      #DECISION CIRCONSTENTIELLES
      #_____________________________________
      #_____________________________________

             #______REGIME LENT____________________
                    if cell.content.v < cell.content.driver.vit_seuil_entree_regime_lent*coefT and distance <
cell.content.driver.dist_seuil_entree_regime_lent and cell.content.v > front_cell.content.v:  #déclenchement de l'état
                          cell.content.etat_c = STOP
                          cell.content.distance_fictive = distance
                          cell.content.TCDC[0] = time.time()
                          cell.content.TCDC[1] =
uniform(cell.content.driver.duree_regime_lent_inf,cell.content.driver.duree_regime_lent_sup)/coefT
#cell.content.driver.periode_stop_and_go

                          #______VOIE GAUCHE REGIME  NORMAL____________________
                    #SI on est sur la voie de droite ET que on est à moins de 2m/s de notre vitesse désirée
                    #ET que le leader va au moins à 1m/s de moins que nous
                    #ET que la distance au leader est plus petite que ce qu'on parcourirais en 6 secondes (v*6)
                    # on dépasserais pas un mec qui est super loin
                    #ET que le véhicule de devant accélère moins que 1 (a < 1)
                    elif (cell.content.voie == 1) and (abs(cell.content.driver.vitesse_voulue*coefT-cell.content.v) >=
cell.content.driver.vitesse_seuil_voie_gauche*coefT) and (cell.content.v-front_cell.content.v) >= -1*coefT and (distance
<= cell.content.v*6/coefT) and (front_cell.content.a < 1*coefT):
                          cell.content.etat_c = VOIE_GAUCHE # voie destination
                          cell.content.driver.vitesse_voulue = max(cell.content.driver.vitesse_voulue,
front_cell.content.v/coefT + uniform(cell.content.driver.vitesse_voie_gauche_inf,
cell.content.driver.vitesse_voie_gauche_sup)) #ou je garde la même ou je vais à celle du mec de devant (+entre 20 et 30)
afin de vouloir doubler
                          cell.content.change = 0 #voie destination : 0
                          cell.content.TCDC[1] = uniform(cell.content.driver.duree_change_voie_gauche_inf,
cell.content.driver.duree_change_voie_gauche_sup)/coefT # durée de l'état : 10 secondes
                          cell.content.TCDC[0] = time.time() # maj de la date de début
```

```
            #______VOIE DROITE REGIME NORMAL_____________________
                  #SI on est sur la voie de Gauche
                  #ET que
                        #OU qu'il y a pas de leader sur l'autre voie
                        #OU qu'on va pas à plus de 1m/s que lui
                        #OU qu'il est plus loin que la distance parcourue en 10 secondes --> rayon de perception :
                        #  pr les cas ou le mec de devant est très loin, ie un mec qu'on atteint en au moins 10
secondes
                        # environ 300 m (gros)
                  elif (cell.content.voie == 0) and ((side_front_cell.content == None) or
((side_front_cell.content.v-min(cell.content.car.limit*coefT * cell.content.driver.vitesse_moyenne,
cell.content.driver.vitesse_voulue*coefT)) >= cell.content.driver.vitesse_seuil_voie_droite*coefT) or (dist_to_side_front
> cell.content.v/coefT*10)):
                        cell.content.etat_c = VOIE_DROITE
#_____________________________________________________________________________________________________________
______
                        cell.content.change = 1
                        cell.content.TCDC[1] = uniform(cell.content.driver.duree_change_voie_droite_inf,
cell.content.driver.duree_change_voie_droite_sup)/coefT
                        cell.content.TCDC[0] = time.time()

            #______ACCELERATION POUR NE PAS ETRE OBSTRUé_____________________NON OBSTRUCTION
                  #SI on est sur la voie de Gauche
                  #ET que la side_front est sur la voie de DROITE
                  #ET que elle veut aller sur la voie de GAUCHE
                  #ET qu'on est à 2 seconde d'elle (2*v)
                  elif (cell.content.voie == 0) and (side_front_cell != None) and (side_front_cell.content.change == 0)
and (side_front_cell.content.voie == 1) and (dist_to_side_front < 2/coefT*cell.content.v) and uniform(0,1) <
cell.content.driver.proba_non_obstruction and cell.content.v - side_front_cell.content.v >
cell.content.driver.seuil_diff_vitesse_NO:
                        cell.content.etat_c = OBSTRUCTION
                        cell.content.driver.vitesse_voulue = min(cell.content.driver.vitesse_voulue +
uniform(cell.content.driver.ajout_vit_voulue_inf,cell.content.driver.ajout_vit_voulue_sup)
,cell.content.car.limit*cell.content.driver.vitesse_sup)
                        temps_dep = (distance + cell.content.v*2/coefT)/cell.content.v
                        cell.content.TCDC[1] = temps_dep/coefT #cette état dure le temps qu'on met à atteindre et à
dépasser de 2 secondes la voiture qui voulait doubler et aller sur la voie de gauche
                        cell.content.TCDC[0] = time.time()


            if cell.content.TCDNC[0] + cell.content.TCDNC[1] < time.time(): #est-ce le temps de changer d'état NON
CIRCONSTENCIEL? si oui allons-y

      #_____________________________
      #_____________________________
      #DECISION NON-CIRCONSTENTIELLES
```

```
	#_____________________________
	#_____________________________

		#_____ACCELERATION_____________________
			indice_de_decision = uniform(0,1) #seuils propres au driver tirés au sort dans un intervalle propre
au profil
			if ((cell.content.etat_c != VOIE_DROITE) or (cell.content.voie == 1)) and indice_de_decision <
cell.content.driver.seuil_indice_ACCEL and cell.content.change == -1:
				cell.content.etat_nc = ACCELERATION
				cell.content.driver.vitesse_voulue = min(cell.content.driver.vitesse_voulue +
uniform(cell.content.driver.ajout_vit_voulue_inf,cell.content.driver.ajout_vit_voulue_sup)
,cell.content.car.limit*cell.content.driver.vitesse_sup) #je veux pas aller plus vite que la vitesse sup
				cell.content.TCDNC[1] = uniform(cell.content.driver.duree_accel_inf,
cell.content.driver.duree_accel_sup)/coefT
				cell.content.TCDNC[0] = time.time()

		#_____RALENTISSEMENT_____________________
			elif (cell.content.etat_c != OBSTRUCTION) and ((cell.content.etat_c != VOIE_GAUCHE) or
(cell.content.voie == 0)) and indice_de_decision < cell.content.driver.seuil_indice_DECEL and cell.content.change == -1:
				cell.content.etat_nc = RALENTISSEMENT
				cell.content.change = -1
				cell.content.driver.vitesse_voulue = max(cell.content.driver.vitesse_voulue -
uniform(cell.content.driver.ajout_vit_voulue_inf,cell.content.driver.ajout_vit_voulue_sup)
,cell.content.car.limit*cell.content.driver.vitesse_inf)
				cell.content.TCDNC[1] = uniform(cell.content.driver.duree_decel_inf,
cell.content.driver.duree_decel_sup)/coefT
				cell.content.TCDNC[0] = time.time()


		#_____NEUTRE_____________________
			else:
				cell.content.etat_nc = NEUTRE
				cell.content.TCDNC[1] = uniform(cell.content.driver.duree_neutre_inf,
cell.content.driver.duree_neutre_sup)/coefT
				cell.content.TCDNC[0] = time.time()
		#_____________________________
		#_____________________________
		#ACTION
		# concerne seulement le stop and go qui a plus d'une phase
		#_____________________________
		#_____________________________Gestion de l'état en cours, or mise à jour, utiliser SLMT par Stop and Go,
car c'est une oscillation gérée ici'
		if cell.content.etat_nc == NEUTRE :
			cell.content.v += uniform(-0.5, 0.5)*coefT
		if cell.content.etat_c == GO :
			if cell.content.v < cell.content.driver.vit_seuil_entree_regime_lent*coefT and distance <
cell.content.driver.dist_seuil_entree_regime_lent and cell.content.v > front_cell.content.v:
```

```
                        cell.content.distance_fictive = distance
                        cell.content.etat_c = STOP

            if cell.content.etat_c == STOP:
                  if distance < cell.content.driver.dist_seuil_entree_regime_lent: #on vérifie que la voiture de devant
est pas trop loin, si elle n'est pas trop loin on s'arrette et on reste à l'arret
                        cell.content.distance_fictive = max(0.1,cell.content.distance_fictive-cell.content.v*dt)
                        distance = cell.content.distance_fictive
                        DV = cell.content.v

                  else: #sinon si elle est trop loin on fait met un temps avant de redémarrer
                        if cell.content.temps_de_reaction_stop_and_go[1] == None:
cell.content.temps_de_reaction_stop_and_go[1] = time.time()
                        elif time.time() < cell.content.temps_de_reaction_stop_and_go[1] +
cell.content.temps_de_reaction_stop_and_go[0]/(front_cell.content.v*8/coefT + 1/coefT):
                        # ~ elif time.time() < cell.content.temps_de_reaction_stop_and_go[1] +
cell.content.temps_de_reaction_stop_and_go[0]:
                              cell.content.distance_fictive = max(0.1,cell.content.distance_fictive-cell.content.v*dt)
                              distance = cell.content.distance_fictive
                              DV = cell.content.v
                        else:#après ce temps, état go
                              cell.content.etat_c = GO
                              cell.content.temps_de_reaction_stop_and_go[1] = None

            return distance,DV

      def update(self,dt):
            for i in range(len(self.voies)):
                  cell = self.voies[i]
                  head = cell
                  while cell.next != head and cell.next != None:
                        front_cell = cell.next
                        #__distance au leader__
                        if front_cell.content.x < cell.content.x:
                              distance = abs(self.taille - cell.content.x -cell.content.car.longueur +
front_cell.content.x)
                              self.voies[i] = front_cell #mAj du plus petit x du peloton (la voiture de x le plus
petit est la tête de la liste chainée)
                        else :
                              distance = abs(front_cell.content.x-cell.content.x-cell.content.car.longueur)
                        if distance < 0.2:
                              self.accidentManager(cell)
                        distance_reelle = distance
                  #_____________________________
                        #___GESTION DES EVENEMENTS___
                        distance,DV = self.gestionaire_evenements(dt,cell,front_cell,distance)
                        #___GESTION DES EVENEMENTS___           peut remplacer l'écart de vitesse ou la distance réelle
```

en une virtuelle

```
                            #______Formule de l'IDM_____________________
                            cell.content.a = self.IDM(cell,front_cell,distance,DV)
                            #______FIN Formule de l'IDM_____________________

                            #______MAJ VITESSE_____________________
                            v = cell.content.v
                            v += cell.content.a*dt
                            #______MAJ VITESSE_____________________

                            #___REACTION OU
NON___#########################################################################################
                            distance = distance_reelle
                            if abs(cell.content.vitesse_passee[0][0] - cell.content.v) >=
cell.content.driver.seuil_reaction*coefT and cell.content.reaction == False and distance > cell.content.v * (2 / coefT):
                                  cell.content.reaction = True # CAS D ENTREE
                                  cell.content.distance_pdt_reaction = distance
                                  cell.content.reaction_date_debut = time.time()
                                  cell.content.driver.temps_reaction = uniform(cell.content.driver.temps_reaction_inf,
cell.content.driver.temps_reaction_sup)/coefT # on retire un nouveau à chaque phases

                            if cell.content.reaction_date_debut + cell.content.driver.temps_reaction < time.time() : #temps
de reaction écoulé, on repasse avec les distances réelles
                                  cell.content.reaction = False # CAS DE SORTIE CAR TEMPS ECOULE

                            if cell.content.reaction == True : # CAS D APPLICATION : si le temps est depassé ALORS on est
passe à false avant, ET si on vient d'y entrer OU que temps pas dépassé, on entre
                                  if distance < cell.content.v*cell.content.driver.temps_reaction :  respect_distance_secu
=distance/cell.content.v*cell.content.driver.temps_reaction
                                  else:  respect_distance_secu = 1
                                  CA = max(0.1,
(abs(cell.content.a-front_cell.content.a)/abs(cell.content.a+front_cell.content.a))**(1/6)) #(8-coefT//2
                                  dfict = (respect_distance_secu * cell.content.distance_pdt_reaction)/CA
                                  cell.content.a = self.IDM(cell,front_cell,max(0.1, dfict),DV) # on refait le calcul de
l'IDM avec une fausse distance
                                  v = cell.content.v
                                  v += cell.content.a*dt
                            #___REACTION OU
NON___#########################################################################################


                      ##########################################################################################
                            cell.content.v = max(0, v)
                            if distance_reelle < 0.6 :
                                  cell.content.v = 0
```

```
                        if cell.content.a < 0 :
                                cell.content.v += uniform(-2, -0.5)*coefT
                        if cell.content.a > 0 :
                                cell.content.v += uniform(0, 1)*coefT
                        cell.content.x = (cell.content.x + cell.content.v*dt) % self.taille
                ##########################################################################################


                        #__calcul de la couleur_____________
                        propLim = (cell.content.v/(self.limitation*coefT))**2
                        cell.content.car.color = (round(((propLim)%1) *250),round((propLim%1)*50) ,
round((abs(1-propLim)%1)*250)   )
                        #__calcul de la couleur_____________



                        #__CHANGEMENT_____________
                        if cell.content.change != -1:
                                side_front_cell = self.find_leader(cell,(cell.content.voie+1)%2)
                                head = self.changeManager(cell,side_front_cell)
                        #__CHANGEMENT_____________



                        #__MESURES_____________
                        if self.mesure_vitesses == True :
                                self.list_vitesses.append(cell.content.v)
                                self.list_distances.append(distance_reelle)
                        #__MESURES_____________


                        #__MEMOIRE_____________
                        if time.time() - cell.content.vitesse_passee[-1][1] > cell.content.passe_temps_avant_mesure :
                                cell.content.vitesse_passee.append((cell.content.v, time.time()))

                        if cell.content.vitesse_passee[0][1] + cell.content.del_vitesse_passe < time.time():
                                del cell.content.vitesse_passee[0]
                        #__MEMOIRE_____________



                        cell = cell.next
```

```
#_____________________________________________________________________________________________
                    if cell.next == head : #cas de la dernière cellule
                          if cell == head:#cas ou la cellule est seule sur sa voie
                                cell.content.a = (1-((cell.content.v)/(cell.content.driver.vitesse_voulue))**4)*
cell.content.car.acceleration_max
                                #______MAJ VITESSE_____________________
                                v = cell.content.v
                                v += cell.content.a*dt
                                #______MAJ VITESSE_____________________
                                distance_reelle = 0
                          else:#cas ou la cellule n'est pas seule sur sa voie et qu'on est en fin de boucle
                                front_cell = cell.next
                                #__distance au leader__
                                if front_cell.content.x < cell.content.x :
                                      distance = abs(self.taille - cell.content.x +
front_cell.content.x-cell.content.car.longueur)
                                      self.voies[i] = front_cell
                                else :
                                      distance = abs(front_cell.content.x-cell.content.x-cell.content.car.longueur)
                                if distance < 0.2:
                                      self.accidentManager(cell)
                                distance_reelle = distance

                                #GESTION D'EVENEMENTS
                                distance,DV = self.gestionaire_evenements(dt,cell,front_cell,distance)

                                #______Formule de l'IDM_____________________
                                cell.content.a = self.IDM(cell,front_cell,distance,DV)
                                #______FIN Formule de l'IDM_____________________


                                #______MAJ VITESSE_____________________
                                v = cell.content.v
                                v += cell.content.a*dt
                                #______MAJ VITESSE_____________________

                                #___REACTION OU
NON___#########################################################################################
                                distance = distance_reelle
                                if abs(cell.content.vitesse_passee[0][0] - cell.content.v) >=
cell.content.driver.seuil_reaction*coefT and cell.content.reaction == False and distance > cell.content.v * (2 / coefT):
                                      cell.content.reaction = True # CAS D ENTREE
                                      cell.content.distance_pdt_reaction = distance
                                      cell.content.reaction_date_debut = time.time()
                                      cell.content.driver.temps_reaction =
uniform(cell.content.driver.temps_reaction_inf, cell.content.driver.temps_reaction_sup)/coefT # on retire un nouveau à
chaque phases
```

```
                                    if cell.content.reaction_date_debut + cell.content.driver.temps_reaction < time.time() :
#temps de reaction écoulé, on repasse avec les distances réelles
                                          cell.content.reaction = False # CAS DE SORTIE CAR TEMPS ECOULE

                                    if cell.content.reaction == True : # CAS D APPLICATION : si le temps est depassé ALORS
on est passe à false avant, ET si on vient d'y entrer OU que temps pas dépassé, on entre
                                          if distance < cell.content.v*cell.content.driver.temps_reaction :
respect_distance_secu =distance/cell.content.v*cell.content.driver.temps_reaction
                                          else:  respect_distance_secu = 1
                                          CA = max(0.1,
(abs(cell.content.a-front_cell.content.a)/abs(cell.content.a+front_cell.content.a))**(1/6)) #(8-coefT//2
                                          dfict = (respect_distance_secu * cell.content.distance_pdt_reaction)/CA
                                          cell.content.a = self.IDM(cell,front_cell,max(0.1, dfict),DV) # on refait le
calcul de l'IDM avec une fausse distance
                                          v = cell.content.v
                                          v += cell.content.a*dt
                              #___REACTION OU
NON___#########################################################################################
                              #___________SORTIE CAS OU HEAD MAIS PAS TOUTE SEULE

                        ##############################################################################################
                              cell.content.v = max(0, v)
                              if distance_reelle < 0.6 :
                                    cell.content.v = 0
                              if cell.content.a < 0 :
                                    cell.content.v += uniform(-2, -0.5)
                              if cell.content.a > 0 :
                                    cell.content.v += uniform(0, 1)
                              cell.content.x = (cell.content.x + cell.content.v*dt) % self.taille
                        ##############################################################################################


                              #__calcul de la couleur____________
                              propLim = (cell.content.v/(self.limitation*coefT))**2
                              cell.content.car.color = (round(((propLim)%1) *250),round((propLim%1)*50) ,
round((abs(1-propLim)%1)*250)   )
                              #__calcul de la couleur____________



                              #__CHANGEMENT____________
                              if cell.content.change != -1:
                                    side_front_cell = self.find_leader(cell,(cell.content.voie+1)%2)
                                    head = self.changeManager(cell,side_front_cell)
                              #__CHANGEMENT____________
```

```
                        #__MESURES_____________
                        if self.mesure_vitesses == True :
                                self.list_vitesses.append(cell.content.v)
                                self.list_distances.append(distance_reelle)
                        #__MESURES_____________


                        #__MEMOIRE_____________
                        if time.time() - cell.content.vitesse_passee[-1][1] > cell.content.passe_temps_avant_mesure :
                                cell.content.vitesse_passee.append((cell.content.v, time.time()))

                        if cell.content.vitesse_passee[0][1] + cell.content.del_vitesse_passe < time.time():
                                del cell.content.vitesse_passee[0]
                        #__MEMOIRE_____________



class Driver:
        def __init__(self,profile : str):
                self.profile = open(profile,'r') #chargement du fichier du profil
                #chargement des données du profil
                self.reaction = int(self.profile.readline())      #seconde
                self.prevoyance = int(self.profile.readline())    #pixel
                self.distance_secu = float(self.profile.readline()) #rapport de la distance de sécu idéale du conducteur sur la
distance de sécu "normale"
                self.vitesse_inf = float(self.profile.readline())   #majore et minore la vitesse voulue rapport de la vitesse inf du
conducteur sur la vitesse max de la route
                self.vitesse_sup = float(self.profile.readline())   #rapport de la vitesse max du conducteur sur la vitesse max de la
route

                self.coeff_distance_secu_inf = float(self.profile.readline())
                self.coeff_distance_secu_sup = float(self.profile.readline())
                self.coeff_acceleration_inf = float(self.profile.readline())
                self.coeff_acceleration_sup = float(self.profile.readline())
                self.coeff_deceleration_inf = float(self.profile.readline())

                self.coeff_deceleration_sup = float(self.profile.readline())
                self.decel_limite_changement_inf = float(self.profile.readline())
                self.decel_limite_changement_sup = float(self.profile.readline())
                self.vit_seuil_entree_regime_lent_inf = float(self.profile.readline())
                self.vit_seuil_entree_regime_lent_sup = float(self.profile.readline())
                self.dist_seuil_entree_regime_lent_inf = float(self.profile.readline())
                self.dist_seuil_entree_regime_lent_sup = float(self.profile.readline())
                self.duree_regime_lent_inf = float(self.profile.readline())
                self.duree_regime_lent_sup = float(self.profile.readline())
                self.duree_change_voie_gauche_inf = float(self.profile.readline())
                self.duree_change_voie_gauche_sup = float(self.profile.readline())
                self.duree_change_voie_droite_inf = float(self.profile.readline())
                self.duree_change_voie_droite_sup = float(self.profile.readline())
                self.duree_accel_inf = float(self.profile.readline())
```

```
                self.duree_accel_sup = float(self.profile.readline())
                self.duree_decel_inf = float(self.profile.readline())
                self.duree_decel_sup = float(self.profile.readline())
                self.duree_neutre_inf = float(self.profile.readline())
                self.duree_neutre_sup = float(self.profile.readline())
                self.vitesse_voie_gauche_inf = float(self.profile.readline())
                self.vitesse_voie_gauche_sup = float(self.profile.readline())
                self.ajout_vit_voulue_inf = float(self.profile.readline())
                self.ajout_vit_voulue_sup = float(self.profile.readline())
                self.seuil_indice_ACCEL = float(self.profile.readline())
                self.seuil_indice_DECEL = float(self.profile.readline())
                self.temps_distance_secu_CV_inf = float(self.profile.readline())
                self.temps_distance_secu_CV_sup = float(self.profile.readline())
                self.vitesse_seuil_voie_droite_inf = float(self.profile.readline())
                self.vitesse_seuil_voie_droite_sup = float(self.profile.readline())
                self.vitesse_seuil_voie_gauche_inf = float(self.profile.readline())
                self.vitesse_seuil_voie_gauche_sup = float(self.profile.readline())
                self.seuil_reaction_inf = float(self.profile.readline())
                self.seuil_reaction_sup = float(self.profile.readline())
                self.temps_reaction_inf = float(self.profile.readline())
                self.temps_reaction_sup = float(self.profile.readline())
                self.proba_non_obstruction = float(self.profile.readline())
                self.seuil_diff_vitesse_NO = float(self.profile.readline())
                #attribus propres au conducteur, déterminés aléatoirement a partir de bornes inf et sup
                self.coeff_distance_secu = uniform(self.coeff_distance_secu_inf,self.coeff_distance_secu_sup)
                self.coeff_acceleration = uniform(self.coeff_acceleration_inf,self.coeff_acceleration_sup)
                self.coeff_deceleration = uniform(self.coeff_deceleration_inf,self.coeff_deceleration_sup)
                self.vitesse_voulue = None
                self.vitesse_moyenne = (self.vitesse_inf+self.vitesse_sup)/2 #centre de l'intervalle précédement défini
                self.decel_limite_changement = uniform(self.decel_limite_changement_inf,self.decel_limite_changement_sup)
                self.vit_seuil_entree_regime_lent = uniform(self.vit_seuil_entree_regime_lent_inf,
self.vit_seuil_entree_regime_lent_sup)
                self.dist_seuil_entree_regime_lent = uniform(self.dist_seuil_entree_regime_lent_inf,
self.dist_seuil_entree_regime_lent_sup)
                self.temps_distance_secu_CV = uniform(self.temps_distance_secu_CV_inf, self.temps_distance_secu_CV_sup)
                self.vitesse_seuil_voie_droite = uniform(self.vitesse_seuil_voie_droite_inf, self.vitesse_seuil_voie_droite_sup)
                self.vitesse_seuil_voie_gauche = uniform(self.vitesse_seuil_voie_gauche_inf, self.vitesse_seuil_voie_gauche_sup)
                self.seuil_reaction = uniform(self.seuil_reaction_inf, self.seuil_reaction_sup)
                self.temps_reaction = uniform(self.temps_reaction_inf, self.temps_reaction_sup)/coefT
class Car:
        def __init__(self,profile : str, color : tuple):
                self.profile = open(profile,'r') #chargement du fichier du profil
                self.color = color
                #chargement des données du profil
                self.acceleration_max = float(self.profile.readline())
                self.freinage_max = float(self.profile.readline())
                self.longueur = int(self.profile.readline())
                self.camion = bool(self.profile.readline())
                self.limit = float(self.profile.readline())


class Cell:
        def __init__(self,driver_profile : str, car_profile : str,name : str):
                self.name = name
                self.x = 0.0
```

```
class Cell:
	def __init__(self,driver_profile : str, car_profile : str,name : str):
		self.name = name
		self.x = 0.0
		self.v = 0.0
		self.a = 0.0
		self.driver = Driver(driver_profile)
		color = (255, 0, 0)
		self.car = Car(car_profile,color)
		self.change = -1
		self.road = None #Road
		self.voie = None #int

		self.etat_c = NEUTRE
		self.etat_nc = NEUTRE
		self.TCDC = [time.time(),0] # Temps Caractéristique de Décision Circonstentielle[Instant de basculement dans l'état,
Temps avant le potentiel changement d'état]
		self.TCDNC = [time.time(),0] # Temps Caractéristique de Décision Non Circonstentielle[Instant de basculement dans
l'état, Temps avant le potentiel changement d'état]
		self.distance_ficitve = None
		self.temps_de_reaction_stop_and_go = [2,None]

		self.passe_temps_avant_mesure = 0.07 #tout les 0.05 on refait une mesure
		self.vitesse_passee = [(self.v, time.time())] #liste des dernières vitesses, de taille 2/0.05 environ
		self.del_vitesse_passe = 2

		self.reaction = None #booléen, en temps de réaction --> true, false : en action, après temps de réaction écoulé
		self.distance_pdt_reaction = None #float, en metre
		self.reaction_date_debut = 0.0 #float, en secondes




	def setRoad(self,road): #méthode qui définit la route de la cellule
		self.road = road  #premet de garder dans un attribut l'information de la route
		self.driver.vitesse_voulue = self.driver.vitesse_moyenne*road.limitation

	def setVoie(self,voie : int): #méthode qui définit la voie de la cellule
		self.voie = voie
		node = DCList(self,None,self.road.voies[voie])
		self.road.voies[voie].previous = node
		self.road.voies[voie] = node

	def __repr__(self):
		return (self.name + " \nposition : " + str(round(self.x,2)) + " m\nspeed : " + str(round(self.v,2)) + "
m/s\nacceleration : " + str(round(self.a,2)) + " m/s²\n")


class Traffic:
	def __init__(self,road):
		self.road = road
		self.cells = []
		fd = open("traffic_param.txt", 'r')
```

```
self.proportion_profil_aggressif = int(fd.readline())
                self.proportion_profil_moyen = int(fd.readline())
                self.proportion_profil_prudent = int(fd.readline())
                self.proportion_depart_voie_gauche = float(fd.readline())
                self.proportion_camion = float(fd.readline())
                self.proportion_lourd = float(fd.readline())



        def generateCells(self,n : int, v_init_inf, v_init_sup, dist_mini = 15):
                nb = int(n*self.proportion_camion)

                taille_slot = (dist_mini*2+4)#taille d'une voiture plus ce qu'elle prend devant a pas derière car une autre le prendre
en compte
                if (n-nb)*taille_slot + nb * (dist_mini*2+18) >= self.road.taille :
                        nb = int((self.road.taille *self.proportion_camion)// (dist_mini*2+18))
                        n = int(self.road.taille/taille_slot)
                print(nb)
                print(n)

                positions = []
                for i in range(n):
                        if i < nb : #quand on pose un camion
                                valid_position = False
                                while not valid_position:
                                        x = randint(0,self.road.taille*0.75)
                                        valid_position = True
                                        for pos in positions:
                                                if (x < pos[0] and x + 17 + dist_mini > pos[0]) or (x > pos[0] and x < pos[0] + dist_mini
+ 17) :
                                                        valid_position = False
                                positions.append((x, 17))

                        else : # quand on pose des voitures, mais on c'est pas si c'est à cote de camion ou de voitures
                                valid_position = False
                                while not valid_position:
                                        x = randint(0,self.road.taille*0.75)
                                        valid_position = True
                                        for pos in positions:
                                                if (x < pos[0] and x + 5 + dist_mini > pos[0]) or (x > pos[0] and x < pos[0] + dist_mini
+ pos[1]):
                                                        valid_position = False

                                positions.append((x, 5))


                        rand_voie = 1
                        if uniform(0, 1) < self.proportion_depart_voie_gauche :
                                rand_voie = 0
                        if uniform(0, 1) < self.proportion_lourd :
                                car = "test_car_lourd.txt"
                        else :
                                car = "test_car.txt"
```

```
                        if i < nb :
                                self.spawnCell(x,rand_voie, "_CAMION", 'test_CAMION.txt', vitesse_init = uniform(v_init_inf,
v_init_sup) )

                        if i < nb + (self.proportion_profil_aggressif/100)*(n-nb) :
                                self.spawnCell(x,0, "_aggressif",vehicule = car,  vitesse_init = min(90, uniform(v_init_inf,
v_init_sup)))

                        elif i < nb + ((self.proportion_profil_moyen+self.proportion_profil_aggressif)/100)*(n-nb) :
                                self.spawnCell(x,rand_voie, "_moyen",vehicule = car,  vitesse_init = uniform(v_init_inf, v_init_sup))

                        elif i < nb +
((self.proportion_profil_moyen+self.proportion_profil_aggressif+self.proportion_profil_moyen)/100)*(n-nb):
                                self.spawnCell(x,rand_voie, "_prudent",vehicule = car, vitesse_init = uniform(v_init_inf, v_init_sup))

        def spawnCell(self,x : float, voie : int, profile : str, vehicule = 'test_car.txt', vitesse_init = 70):
                        cell = Cell('test_driver'+ profile +'.txt',vehicule,'test_cell' + str(len(self.cells)))
                        self.cells.append(cell)
                        cell.setRoad(self.road)
                        cell.voie = voie #randint(0,len(self.road.voies)-1)
                        cell.x = x
                        cell.v = (vitesse_init/3.6)*coefT
                        back_cell = self.road.voies[cell.voie]
                        head = self.road.voies[cell.voie]

                        if back_cell.content == None: # liste vide, rien sur la route, on initialise

                                back_cell.content = cell
                                back_cell.previous = back_cell
                                back_cell.next = back_cell

                        elif back_cell.content.x > x :
                                front_cell = self.road.voies[cell.voie]
                                back_cell = front_cell.previous
                                node = DCList(cell,back_cell,front_cell)
                                back_cell.next = node
                                front_cell.previous = node
                                self.road.voies[cell.voie] = node
                        else:
                                while back_cell.next != head and back_cell.next.content.x < x:
                                        back_cell = back_cell.next
                                front_cell = back_cell.next
                                node = DCList(cell,back_cell,front_cell)
                                back_cell.next = node
                                front_cell.previous = node
                        return cell
```

```
                        back_cell = self.road.voies[cell.voie]
                        head = self.road.voies[cell.voie]

                        if back_cell.content == None: # liste vide, rien sur la route, on initialise

                                back_cell.content = cell
                                back_cell.previous = back_cell
                                back_cell.next = back_cell

                        elif back_cell.content.x > x :
                                front_cell = self.road.voies[cell.voie]
                                back_cell = front_cell.previous
                                node = DCList(cell,back_cell,front_cell)
                                back_cell.next = node
                                front_cell.previous = node
                                self.road.voies[cell.voie] = node
                        else:
                                while back_cell.next != head and back_cell.next.content.x < x:
                                        back_cell = back_cell.next
                                front_cell = back_cell.next
                                node = DCList(cell,back_cell,front_cell)
                                back_cell.next = node
                                front_cell.previous = node
                        return cell

NEUTRE = -1
GO = 0
STOP = 1
RALENTISSEMENT = 2
ACCELERATION = 3
VOIE_GAUCHE = 4
VOIE_DROITE = 5

import win32gui,win32api,win32con
import time
from random import randint
from doubleChainedList import DCList
from classes import Road,Driver,Car,Cell,Traffic
import os
```

```
import pygame
from pygame.locals import *
from display import displayCalculs, drawRoad, drawCell

#_____________________________________CARACTERISTIQUES FENETRE PYGAME
HEIGHT,WIDTH = 840,1530
dim = (HEIGHT, WIDTH)
BACKGROUND = (50,50,50) #couleur du fond
WINDOW = pygame.display.set_mode((WIDTH, HEIGHT)) #Setting WINDOW
pygame.display.set_caption('Traffic SIM') #Window Name
WINDOW.fill(BACKGROUND)
pygame.init()
#_____________________________________


def main():
        run = True
        road = Road('test road.txt') #objet route
        traffic = Traffic(road)
        timestamp = time.time()
        CLOCK = pygame.time.Clock()
        drawRoad(road, WINDOW, dim)

        taillePort, nbPortions, gap, roadTness, tness = displayCalculs(road, dim)

        cell = None
        while run:
                dt = time.time()-timestamp
                timestamp = time.time()
                road.update(dt)
                # ~ if win32api.GetKeyState(0x1B) < 0:       run = False #échap met fin a la boucle while sans fermer la fenêtre

                #_____________________________________AFFICHAGE
                CLOCK.tick(60)
                drawRoad(road, WINDOW, dim)
                drawCell(road, WINDOW, dim)

                for event in pygame.event.get():
                        if event.type == pygame.QUIT:
                          run = False
```

```
                if event.type == pygame.MOUSEBUTTONDOWN: #dynamic cell spawner

                        a,b = pygame.mouse.get_pos()
                        indPort = (b-5)//(roadTness+gap)
                        v = ((b-5)-indPort*(roadTness+gap))//tness
                        if v >= len(road.voies) :
                                v = v//4
                        x = (a/WIDTH)*taillePort + indPort*taillePort
                        cell = traffic.spawnCell(x, v)

        pressed = pygame.key.get_pressed()
        if pressed[pygame.K_ESCAPE]:
                run = False

        if pressed[pygame.K_RIGHT]:
                cell.v += 1
                # ~ cell.driver.vitesse_voulue += 1
        if pressed[pygame.K_LEFT]:
                cell.v -= 1
                # ~ cell.driver.vitesse_voulue -= 1
        if pressed[pygame.K_DOWN]:
                cell.change = 1
        if pressed[pygame.K_UP]:
                cell.change = 0
        # ~ if pressed[pygame.K_a]:
                # ~ cell.change = 2
        pygame.display.update()
        #_____________________________________FIN AFFICHAGE

if __name__ == '__main__':
        main()

from simulation_graphics import main_nodisplay
import matplotlib.pyplot as plt
from classes import coefT
c = coefT
nb_exp = 10
```

```
def launch():
        fp = open("mes"+str(c)+".txt", "w")
        fp.close()
        fp = open("mes"+str(c)+".txt", "w")


        for j in range(nb_exp):
                T, X = main_nodisplay(0)
                fp.write(str(T))
                fp.write("\n")
                fp.write(str(X))
                fp.write("\n")
                print(c)
        return
#_____________________________________
#_____________________________________
def drawing():
        colors = ['b','g','r','c','m','y','k','w', 'r', 'r', 'b', 'c', 'b', 'c', 'c', 'c', 'c', 'c', 'c']
        for i in [1, 4,6, 8, 10, 16]:
                fp = open("mes"+str(i)+".txt", "r")
                for j in range(nb_exp-1): #parcours des mesures à coef fixé
                        T = eval([e.strip() for e in fp.readline().splitlines()][0])
                        X = eval([e.strip() for e in fp.readline().splitlines()][0])
                        plt.plot(T, X, color = colors[i] )
                T = eval([e.strip() for e in fp.readline().splitlines()][0])
                X = eval([e.strip() for e in fp.readline().splitlines()][0])
                plt.plot(T, X, color = colors[i], label = "coef = "+str(i))
                fp.close()
        return
# ~ fp = open("mes"+str(1)+".txt", "r")
# ~ T = eval([e.strip() for e in fp.readline().splitlines()][0])
# ~ X = eval([e.strip() for e in fp.readline().splitlines()][0])
# ~ print(T)
# ~ print("\n et \n")
# ~ print(X)
# Ajouter des étiquettes et un titre

# ~ launch()
drawing()
plt.xlabel("x(t)")
plt.ylabel("t")
plt.title('position en x selon le temps pour un facteur ')
plt.legend()

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