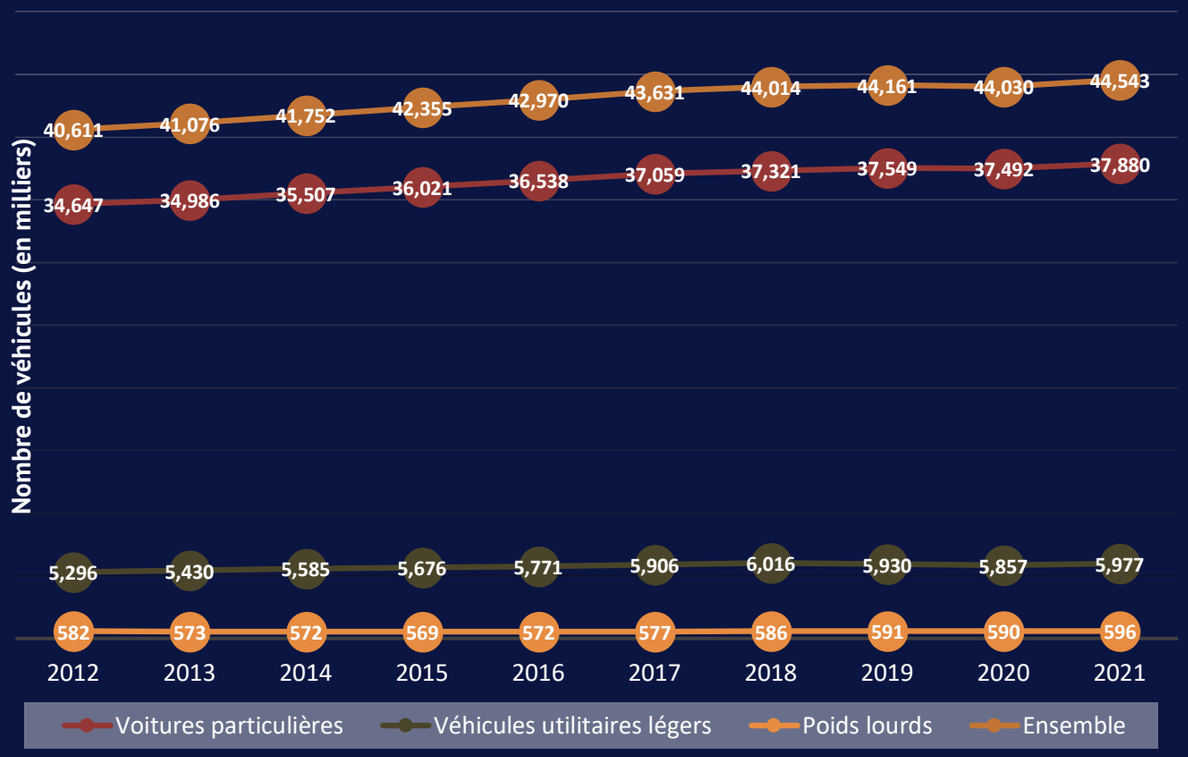
MODELISATION ET OPTIMISATION DU TRAFIC ROUTIER

LA VILLE

KEMMOU Majda
15490

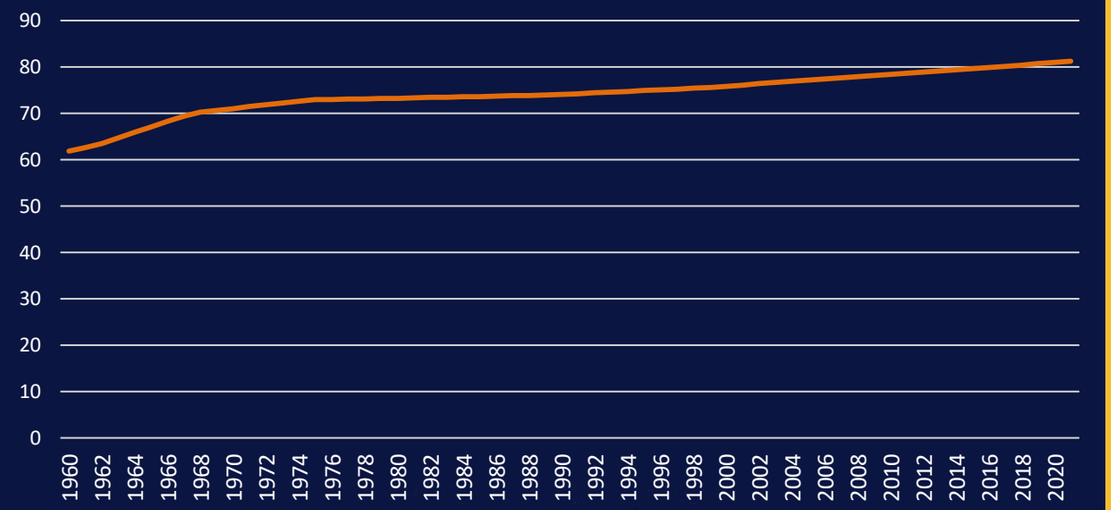
QUELQUES CHIFFRES POUR COMMENCER

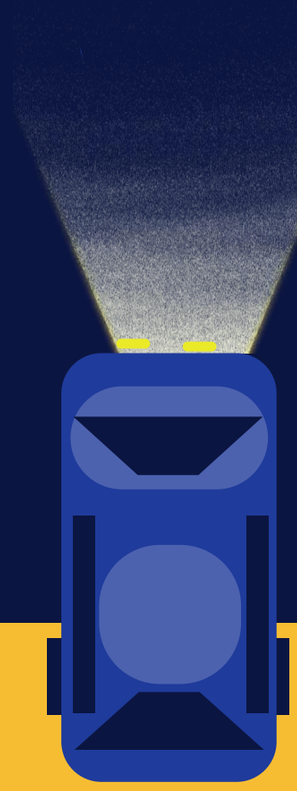
Nombre de véhicules en circulation en France en fonction du temps



Source : Insee

Evolution du pourcentage de la population urbaine française en fonction du temps





Peut-on mettre en évidence des moyens de fluidifier le trafic ?

1. MODELISATION
2. SIMULATION / RESULTATS
3. PARADOXE DE BRAESS

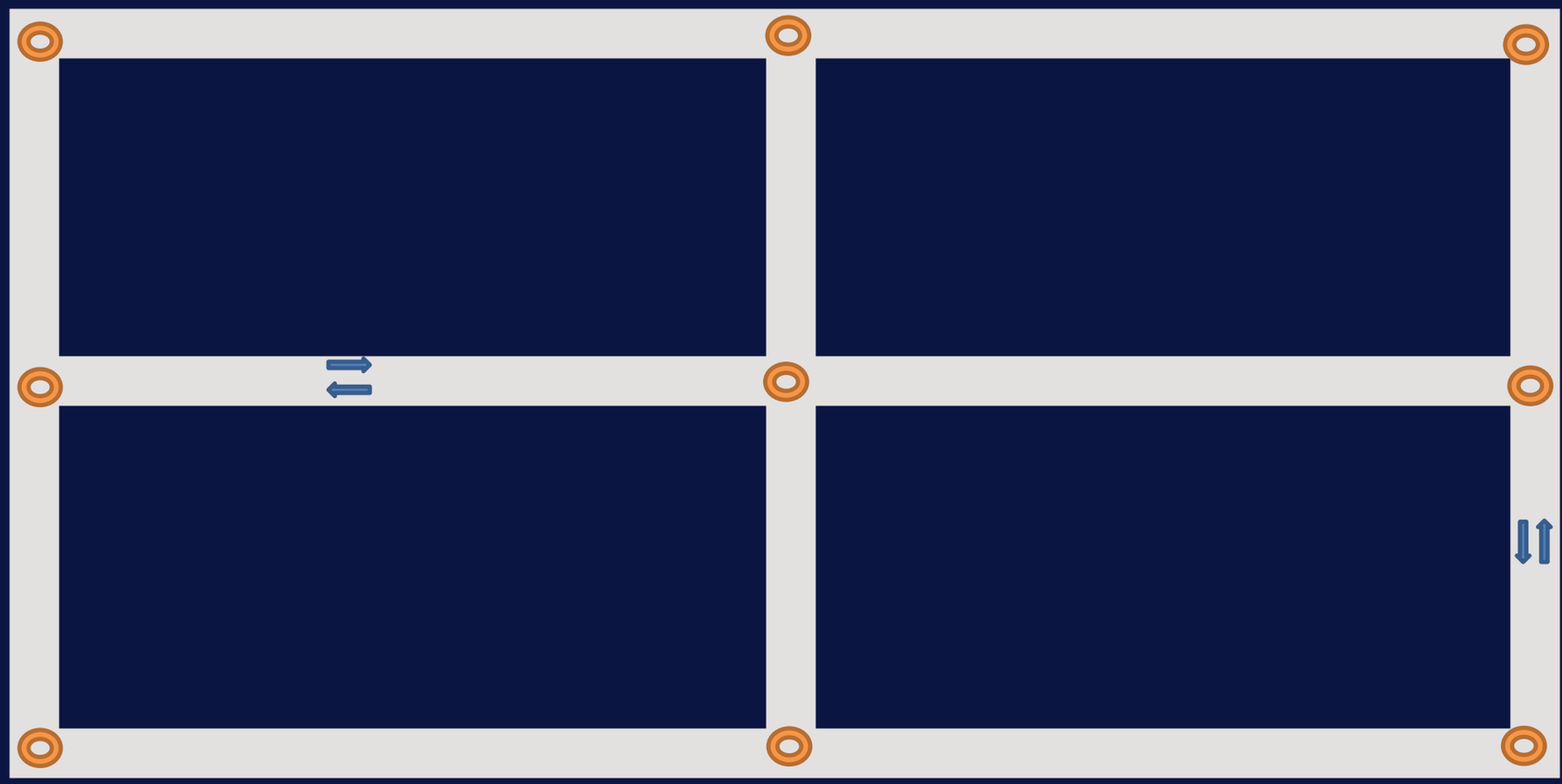
1 - MODELISATION

1 MODELISATION

2 SIMULATION / RESULTATS

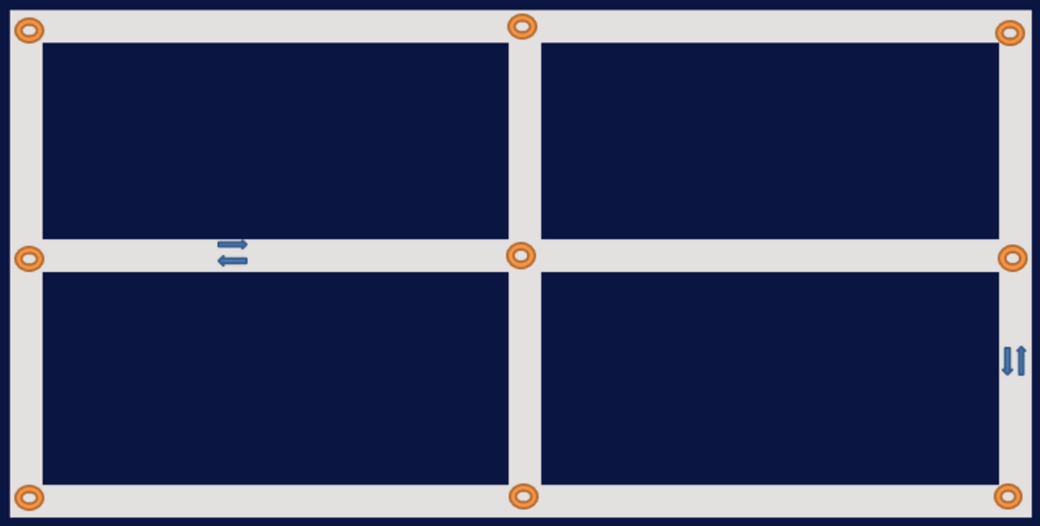
3 PARADOXE DE BRAESS

MODELISATION D'UNE PORTION DE VILLE



GESTION DES INTERSECTIONS

(MATRICE DE PROBABILITES CUMULEES)



```
[[0.  0.  0.5 0.  0.  0.  0.5 0.  0. ]
 [0.  0.  0.  0.5 0.  0.  0.5 0.  0. ]
 [0.5 0.  0.  0.  0.3 0.  0.  0.  0.2]
 [0.  0.3 0.  0.  0.  0.4 0.  0.  0.3]
 [0.  0.  0.5 0.  0.  0.  0.  0.5 0. ]
 [0.  0.  0.  0.5 0.  0.  0.  0.5 0. ]
 [0.4 0.4 0.  0.  0.  0.  0.  0.  0.2]
 [0.  0.  0.  0.  0.3 0.3 0.  0.  0.4]
 [0.  0.  0.2 0.4 0.  0.  0.2 0.2 0. ]]
```

... vers l'intersection j.

De l'intersection i ... ➡

```
[[0.  0.  0.5 0.5 0.5 0.5 1.  1.  1. ]
 [0.  0.  0.  0.5 0.5 0.5 1.  1.  1. ]
 [0.5 0.5 0.5 0.5 0.8 0.8 0.8 0.8 1. ]
 [0.  0.3 0.3 0.3 0.3 0.7 0.7 0.7 1. ]
 [0.  0.  0.5 0.5 0.5 0.5 0.5 1.  1. ]
 [0.  0.  0.  0.5 0.5 0.5 0.5 1.  1. ]
 [0.4 0.8 0.8 0.8 0.8 0.8 0.8 0.8 1. ]
 [0.  0.  0.  0.  0.3 0.6 0.6 0.6 1. ]
 [0.  0.  0.2 0.6 0.6 0.6 0.8 1.  1. ]]
```

L'AUTOMATE CELLULAIRE

basé sur le modèle de NaSch

1. La voiture accélère si elle n'a pas encore atteint la vitesse maximale.
2. La voiture décélère si la distance avec la voiture suivante ne suffit pas.
3. La voiture freine avec une probabilité de p.
4. Les véhicules avancent.

L'INTELLIGENT DRIVER MODEL

(IDM)

- **Distance de sécurité :**

$$s^*(v,\Delta v) = s_0 + \max\left\{0, vT + \frac{v\Delta v}{2\sqrt{ab}}\right\}$$

- **Accélération décroissante de la vitesse, croissante de la distance inter-véhicule :**

$$\frac{dv}{dt} = a\left[1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{s^*(v,\Delta v)}{s}\right)^2\right]$$

- **a accélération maximale**
- **b décélération confortable**
- **s distance leader-follower**
- **v_0 vitesse maximale**
- **s_0 distance de sécurité minimale**
- **Δv différence de vélocité leader-follower**
- **δ exposant d'accélération**
- **T délai de sécurité**

METHODE DE RESOLUTION

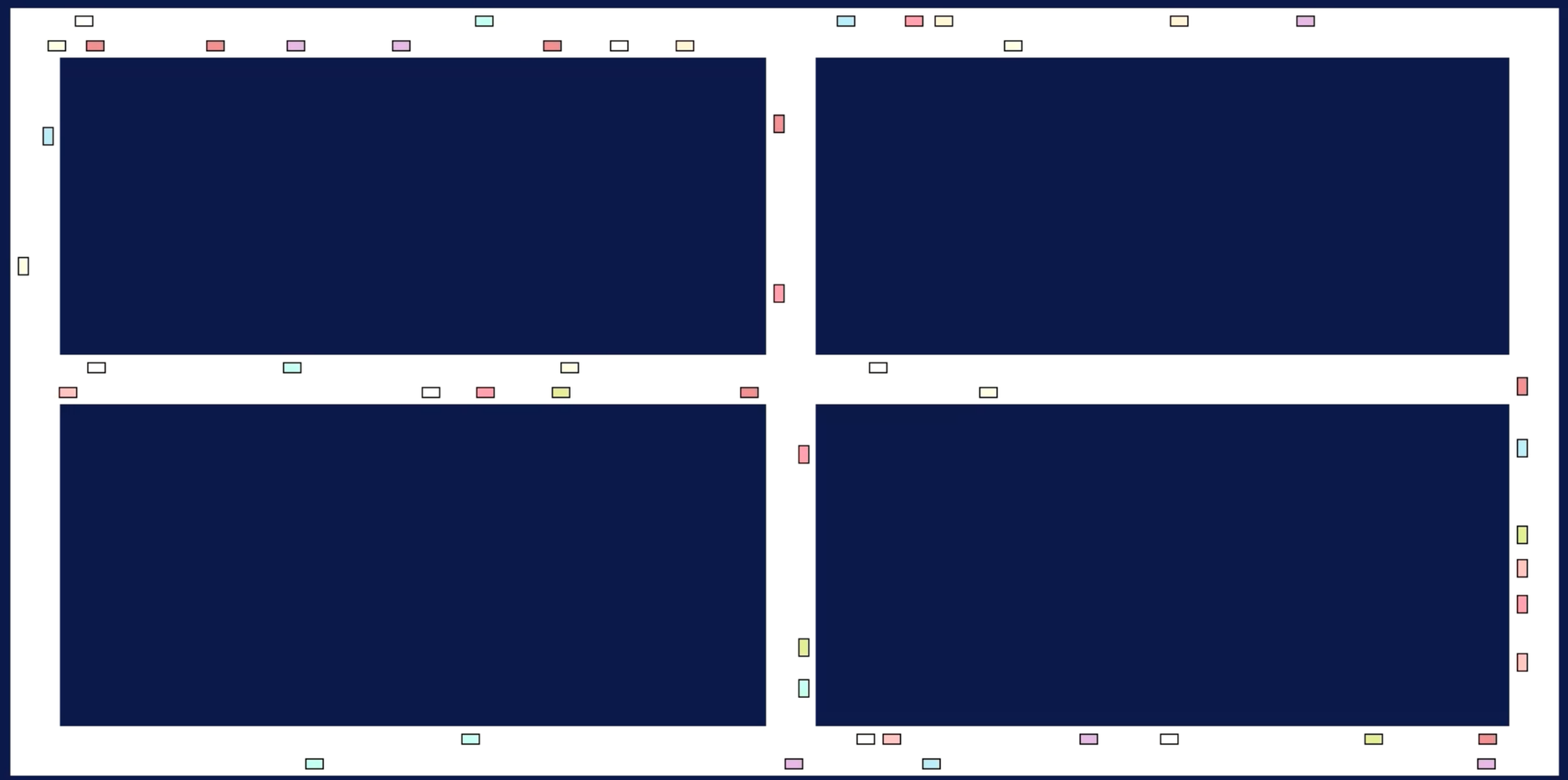
(METHODE D'EULER)

$$\frac{dv}{dt} = a\left[1-\left(\frac{v}{v_0}\right)^{\delta}-\left(\frac{s^*(v,\Delta v)}{s}\right)^2\right]$$

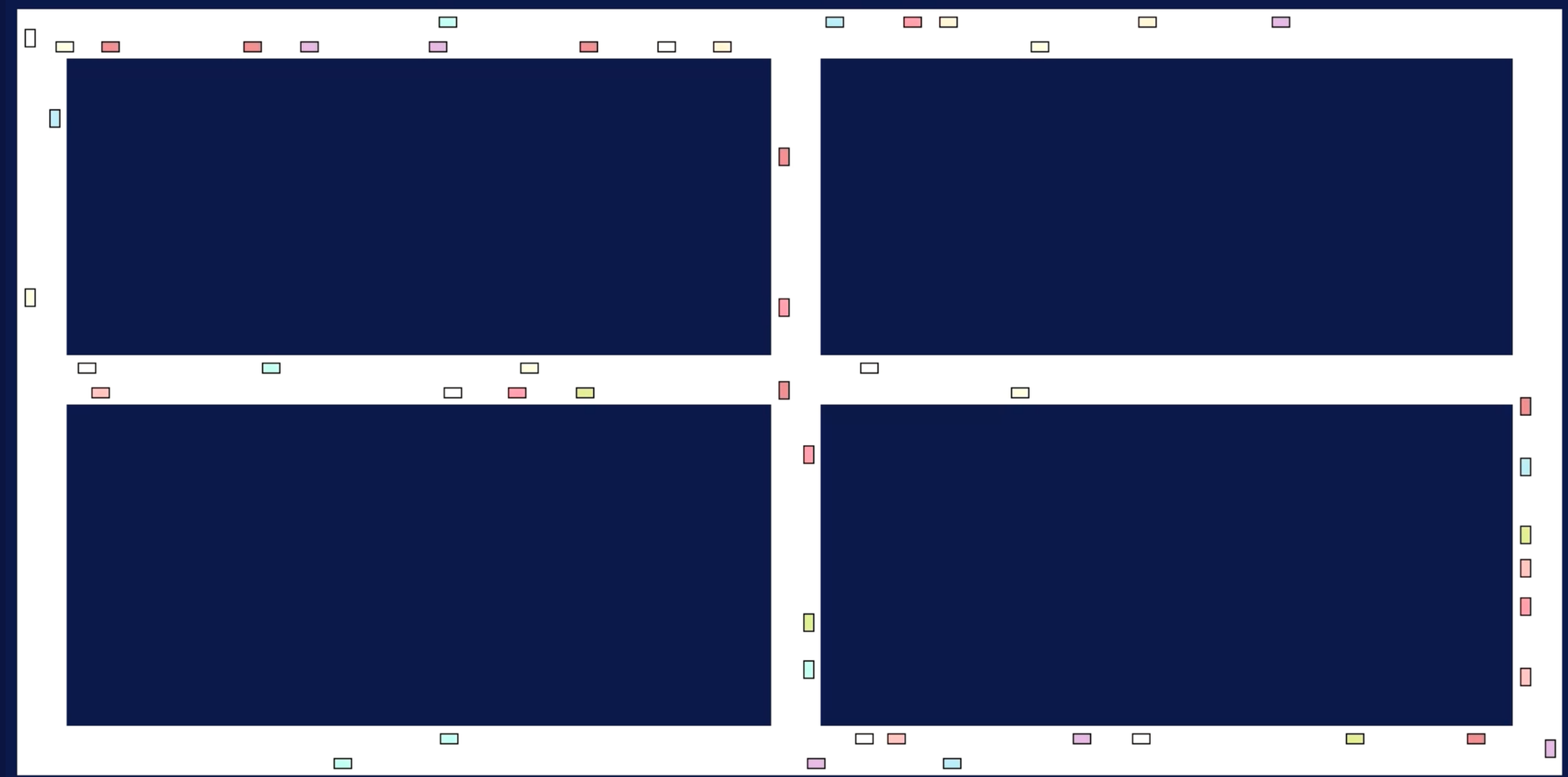
- $\frac{dv}{dt}(t) \approx \frac{v(t+\Delta t)-v(t)}{\Delta t}$ donc $\begin{cases} v(t+\Delta t)=v(t)+\Delta t.a(t) \\ x(t+\Delta t)=x(t)+\frac{1}{2}a(t)\Delta t^2 \end{cases}$

2 – SIMULATION & RESULTATS

SIMULATION POUR 50 VEHICULES



SIMULATION POUR 50 VEHICULES

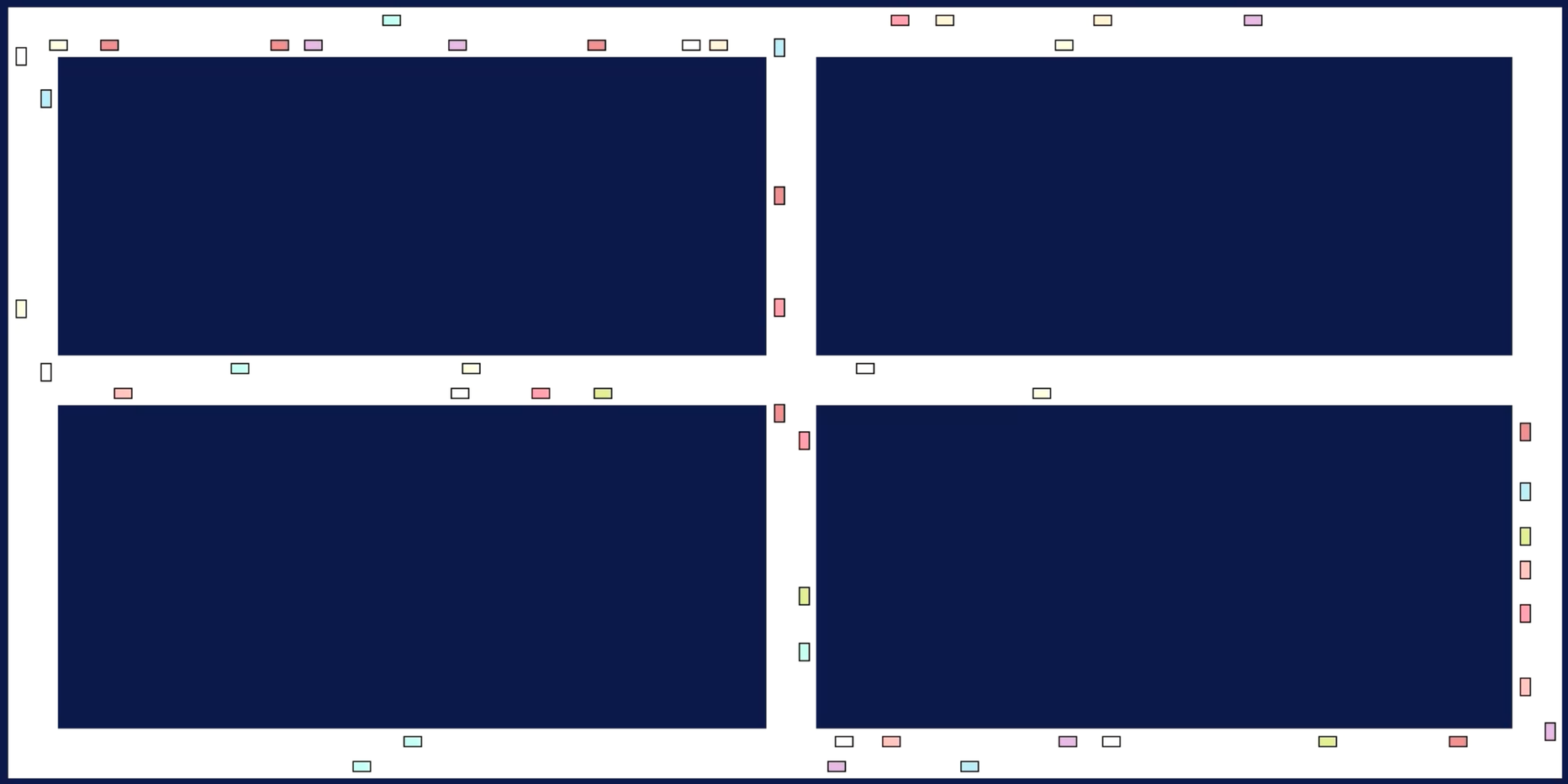


1 MODELISATION

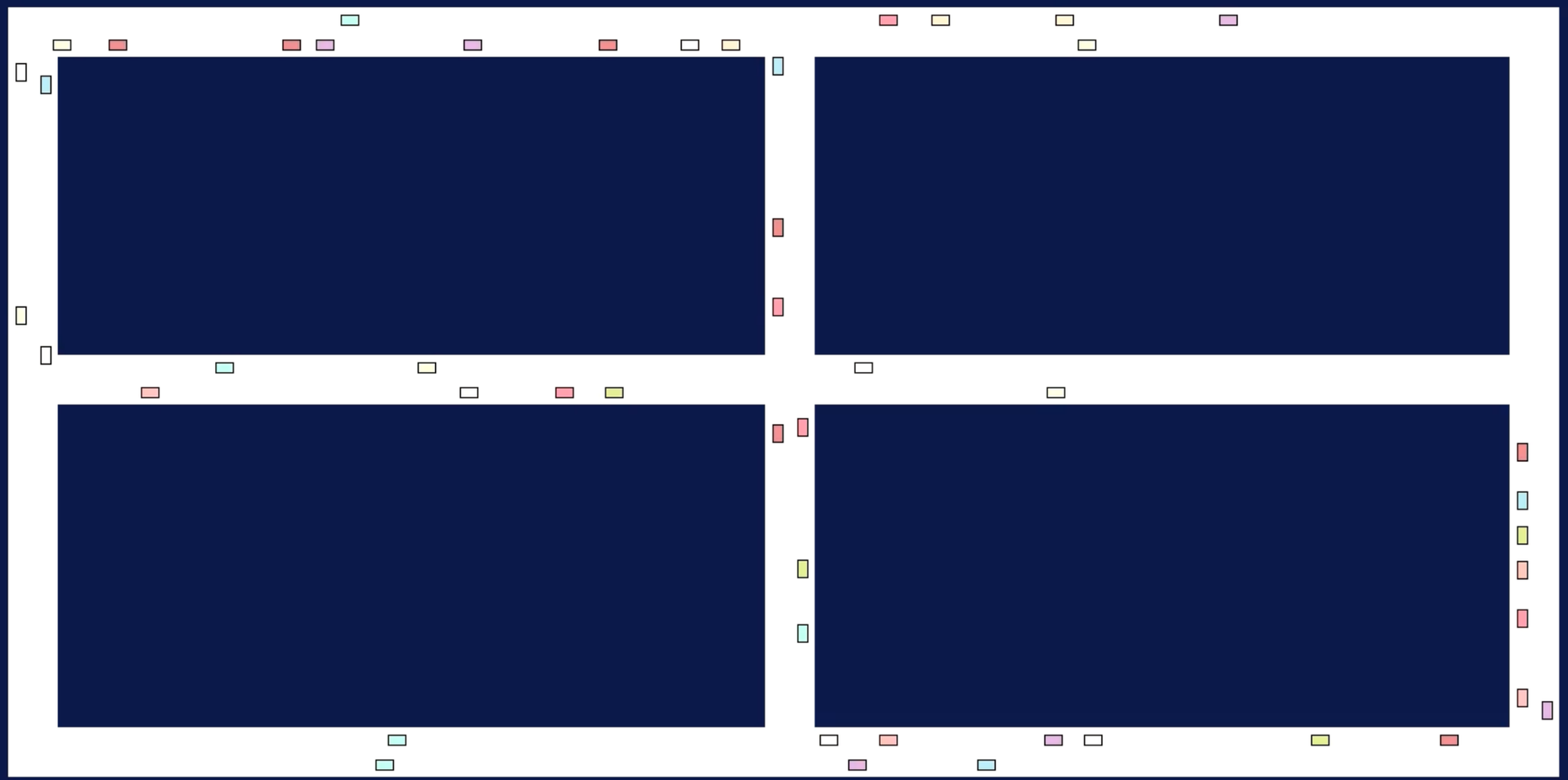
2 SIMULATION / RESULTATS

3 PARADOXE DE BRAESS

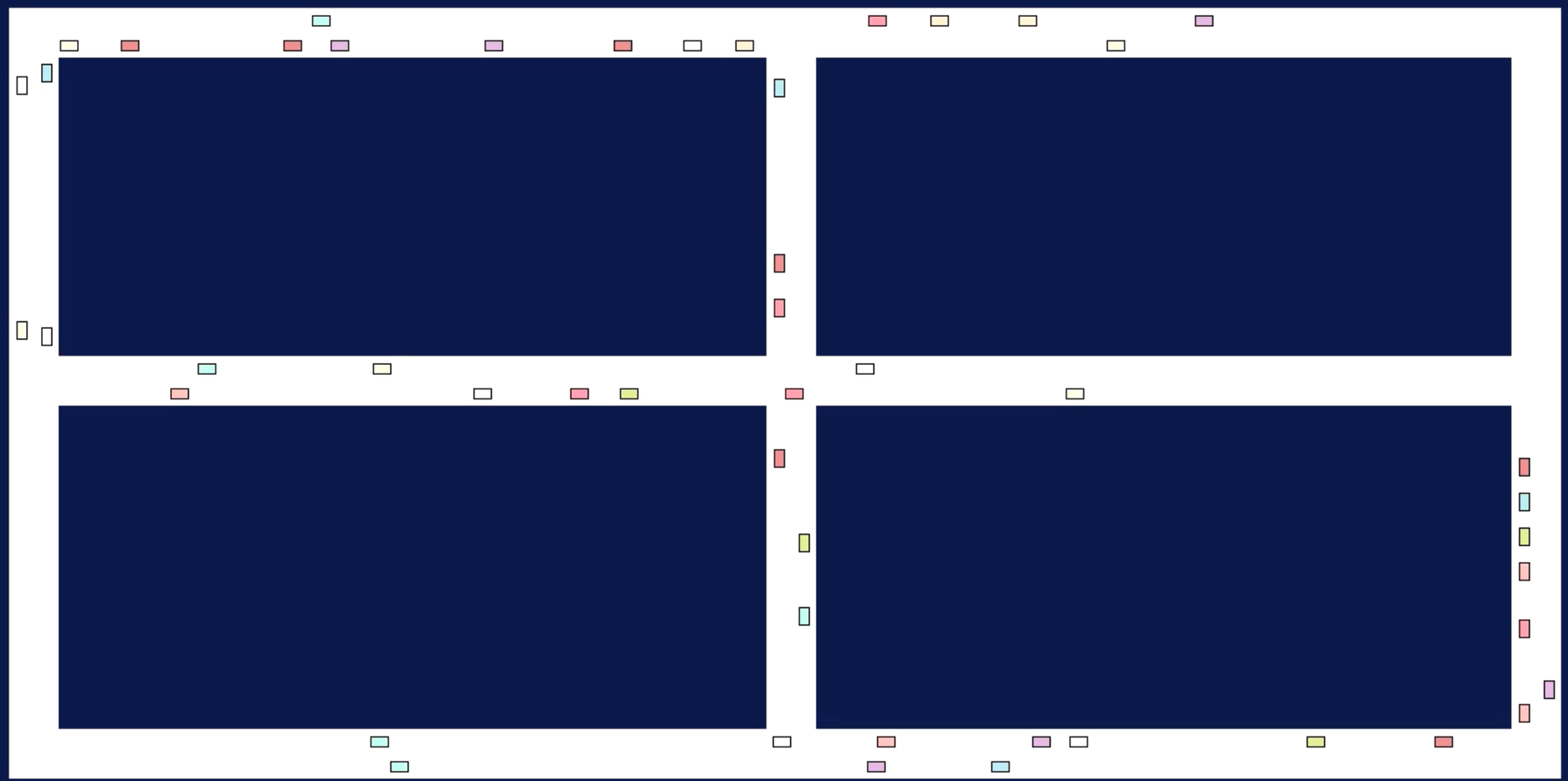
SIMULATION POUR 50 VEHICULES



SIMULATION POUR 50 VEHICULES



SIMULATION POUR 50 VEHICULES

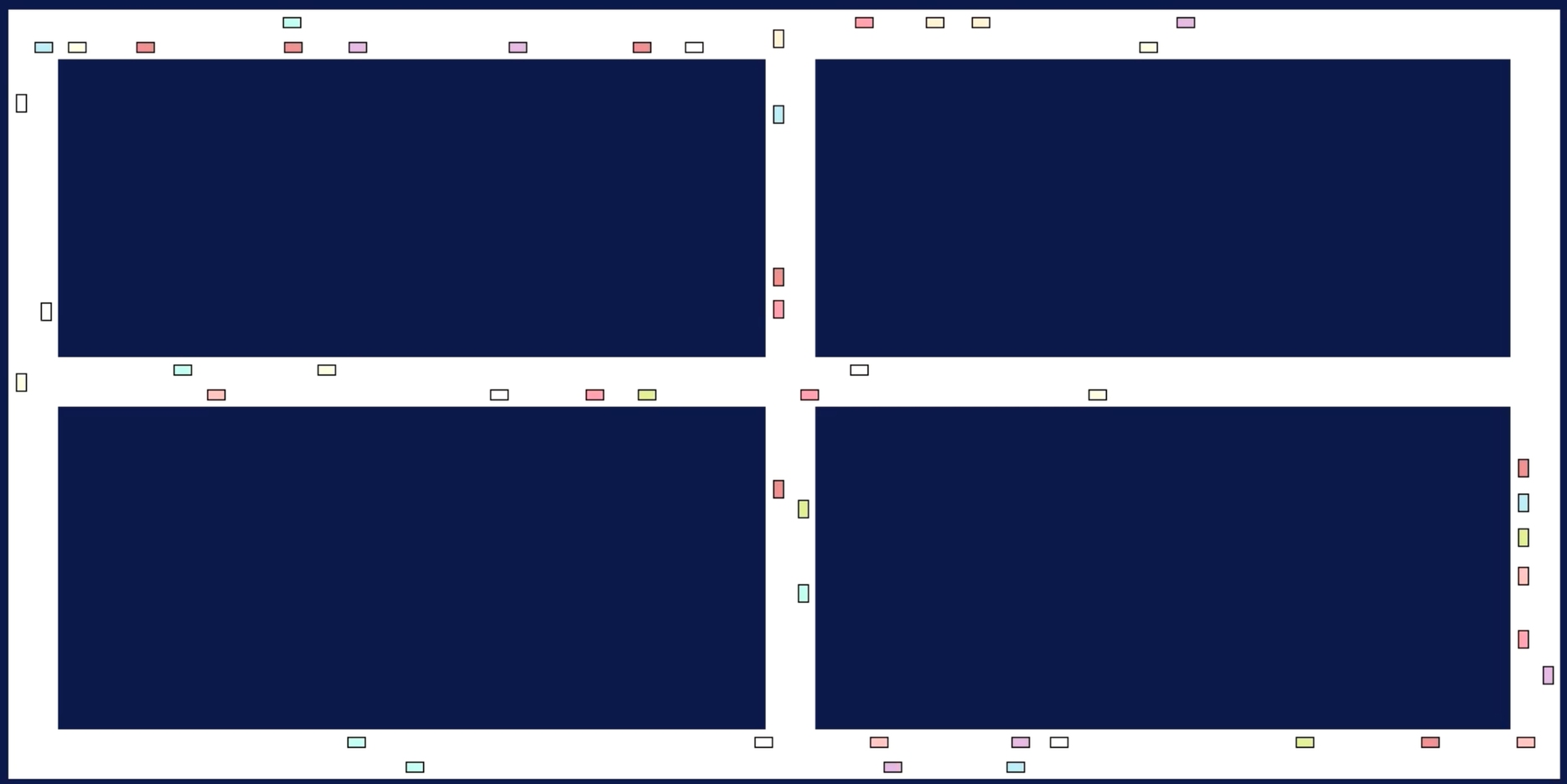


1 MODELISATION

2 SIMULATION / RESULTATS

3 PARADOXE DE BRAESS

SIMULATION POUR 50 VEHICULES

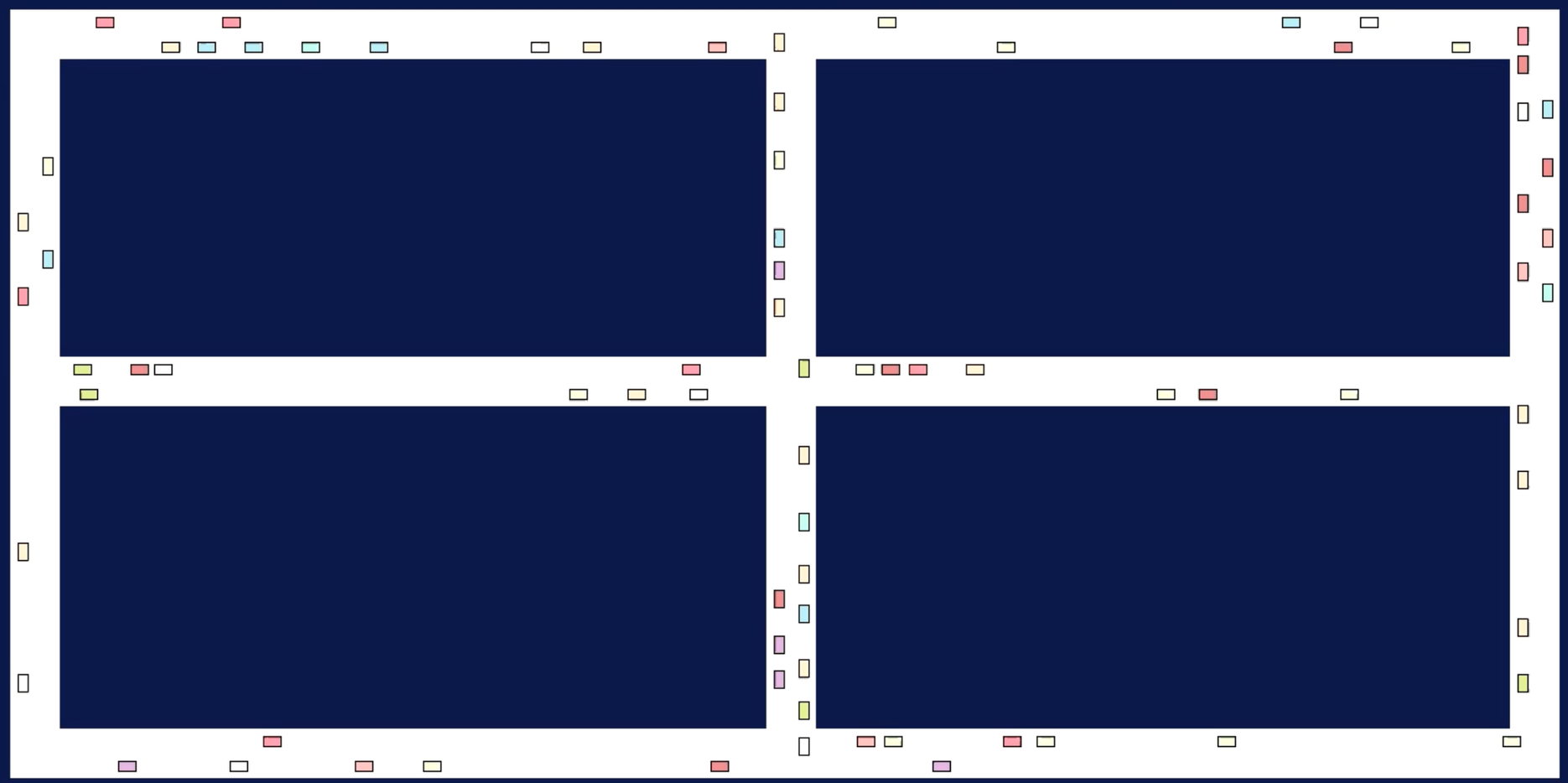


1 MODELISATION

2 SIMULATION / RESULTATS

3 PARADOXE DE BRAESS

SIMULATION POUR 80 VEHICULES

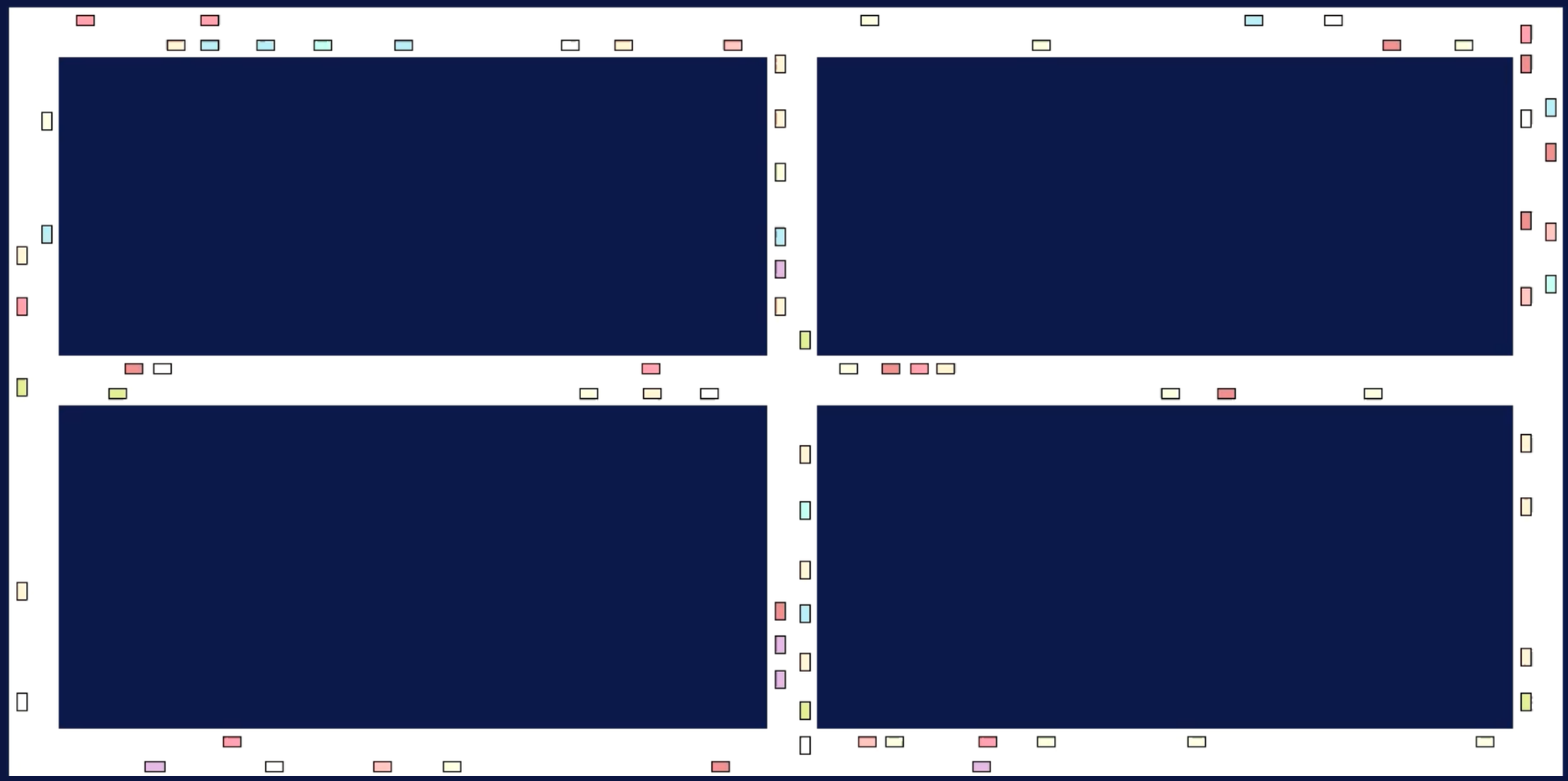


1 MODELISATION

2 SIMULATION / RESULTATS

3 PARADOXE DE BRAESS

SIMULATION POUR 80 VEHICULES

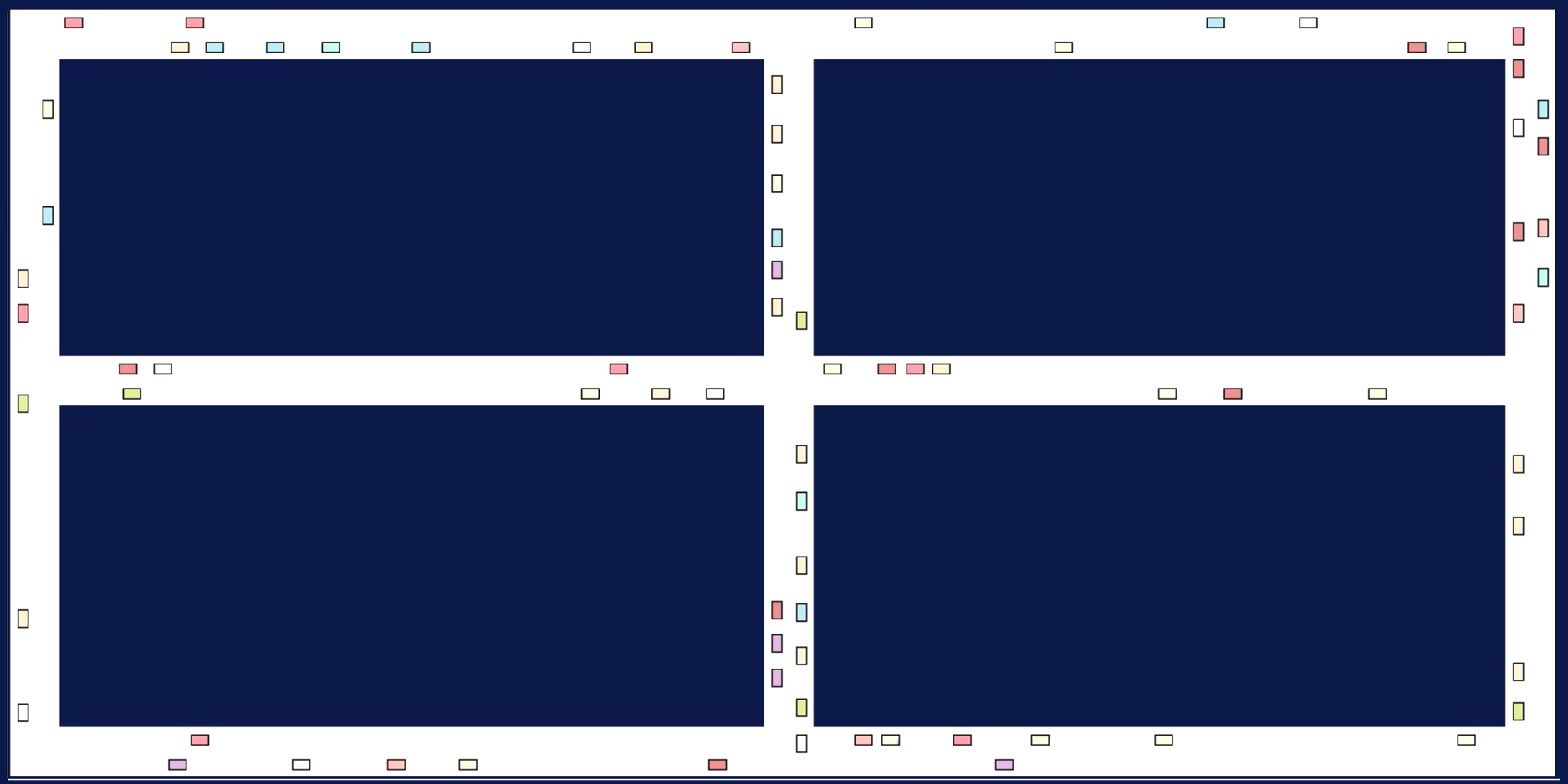


1 MODELISATION

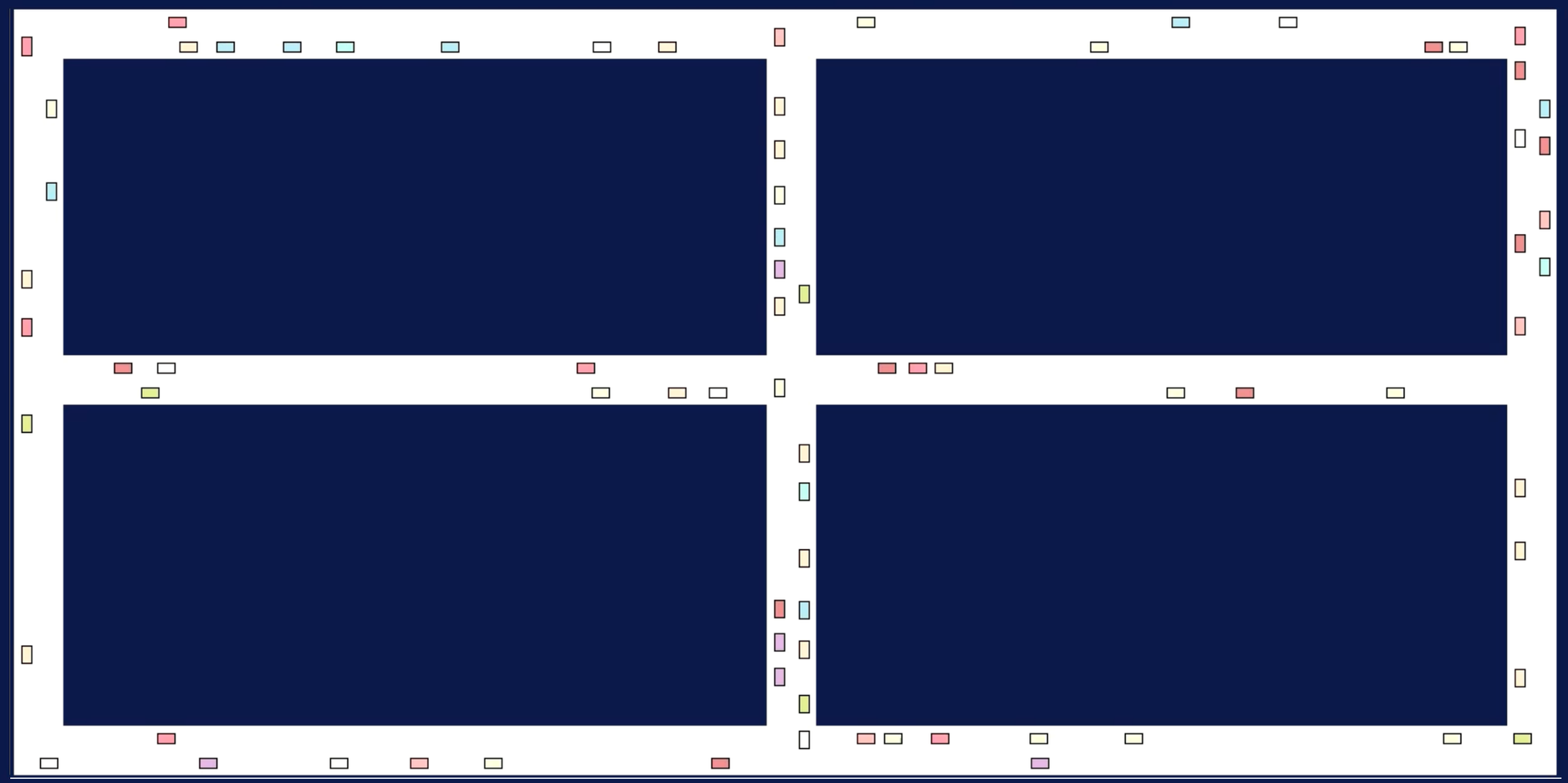
2 SIMULATION / RESULTATS

3 PARADOXE DE BRAESS

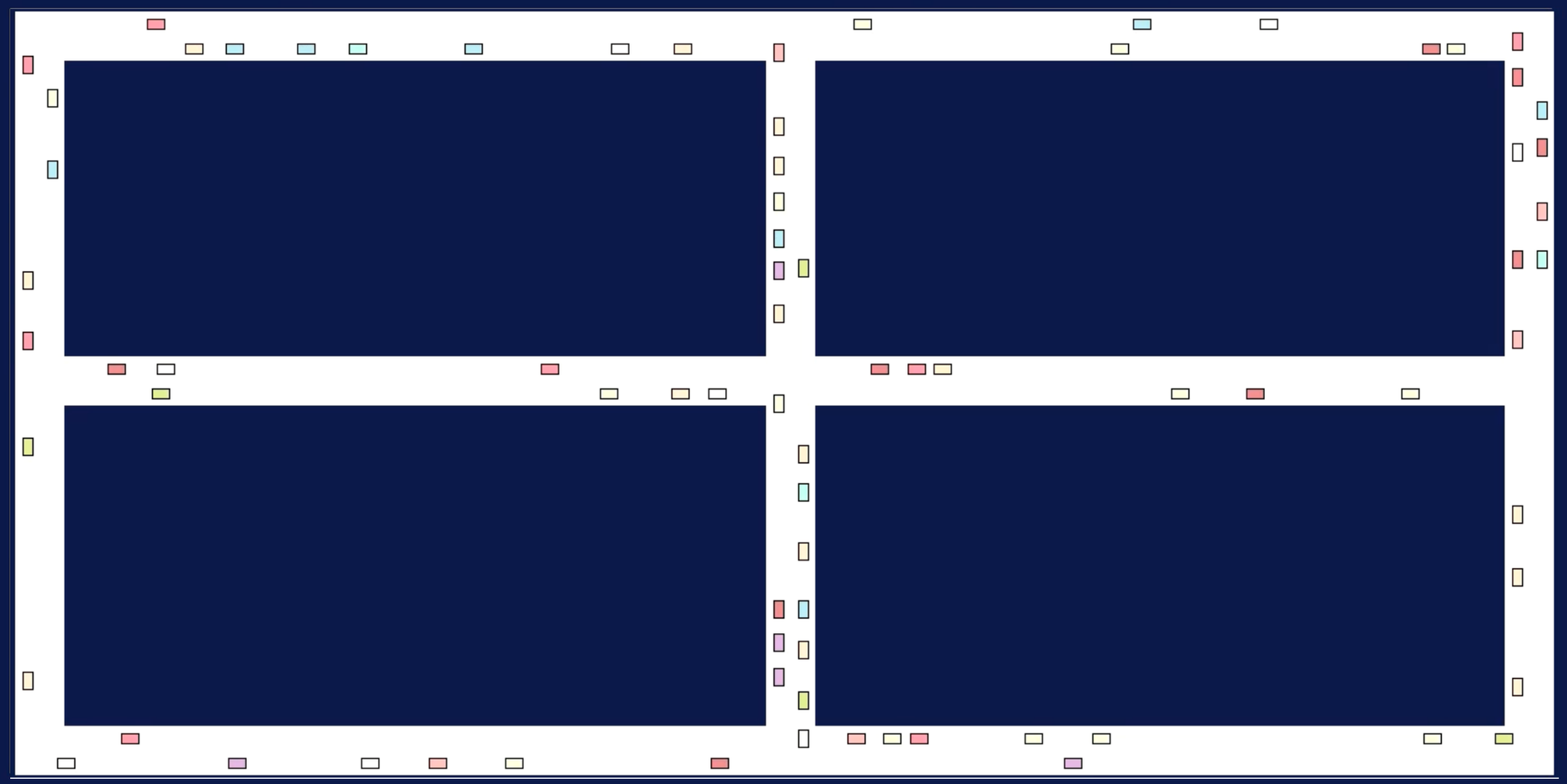
SIMULATION POUR 80 VEHICULES



SIMULATION POUR 80 VEHICULES



SIMULATION POUR 80 VEHICULES

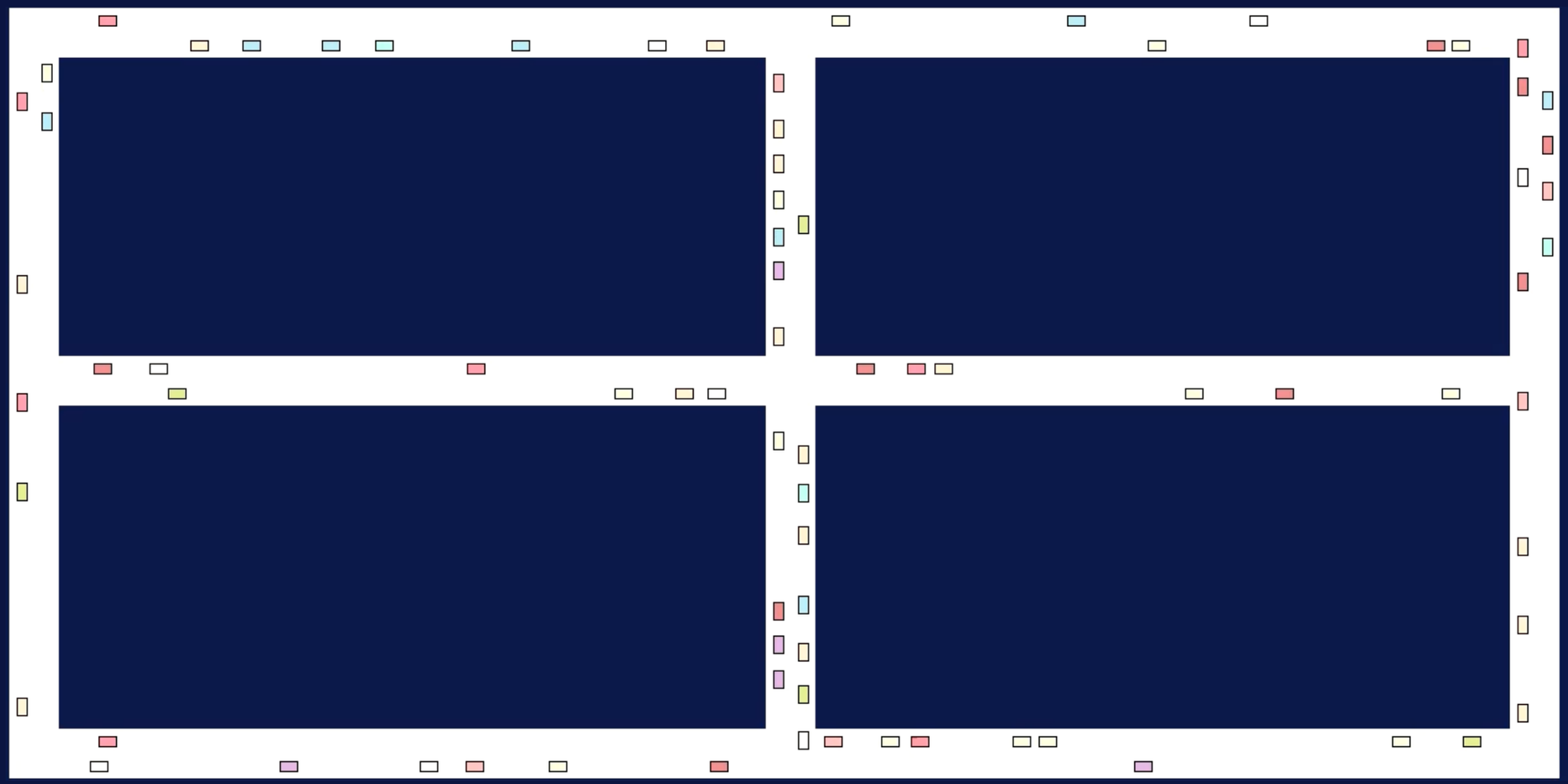


1 MODELISATION

2 SIMULATION / RESULTATS

3 PARADOXE DE BRAESS

SIMULATION POUR 80 VEHICULES

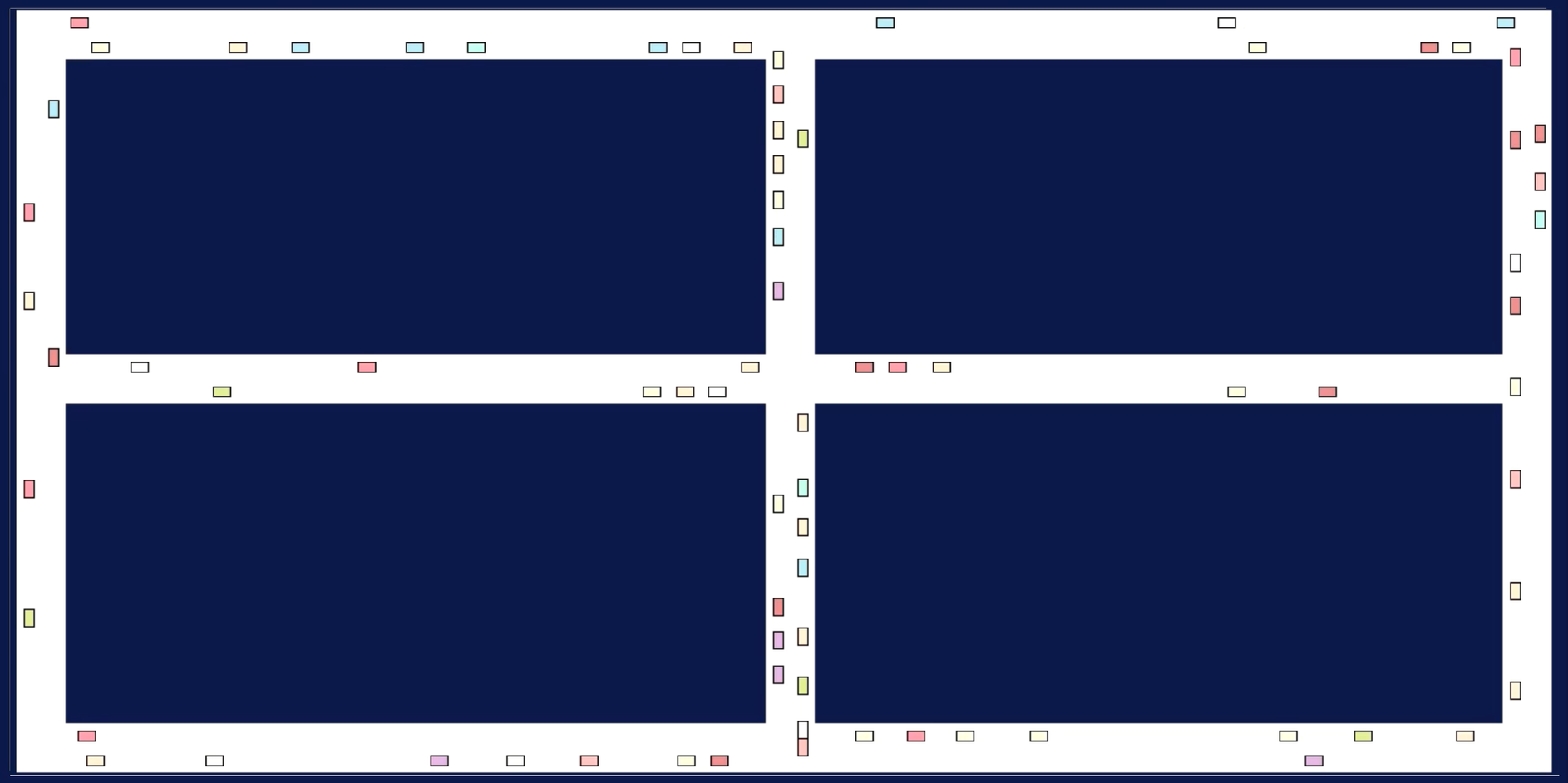


1 MODELISATION

2 SIMULATION / RESULTATS

3 PARADOXE DE BRAESS

SIMULATION POUR 80 VEHICULES

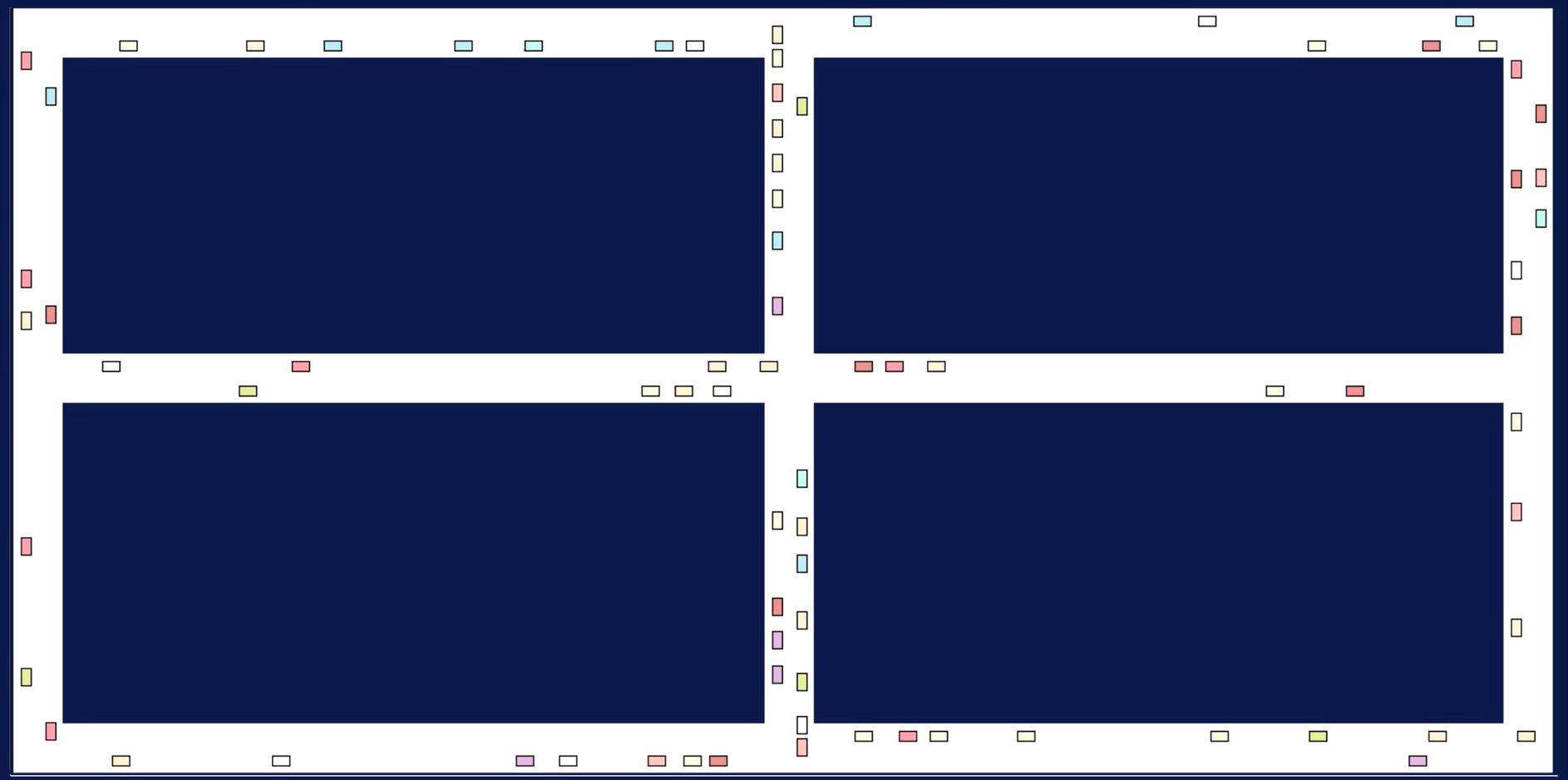


1 MODELISATION

2 SIMULATION / RESULTATS

3 PARADOXE DE BRAESS

SIMULATION POUR 80 VEHICULES

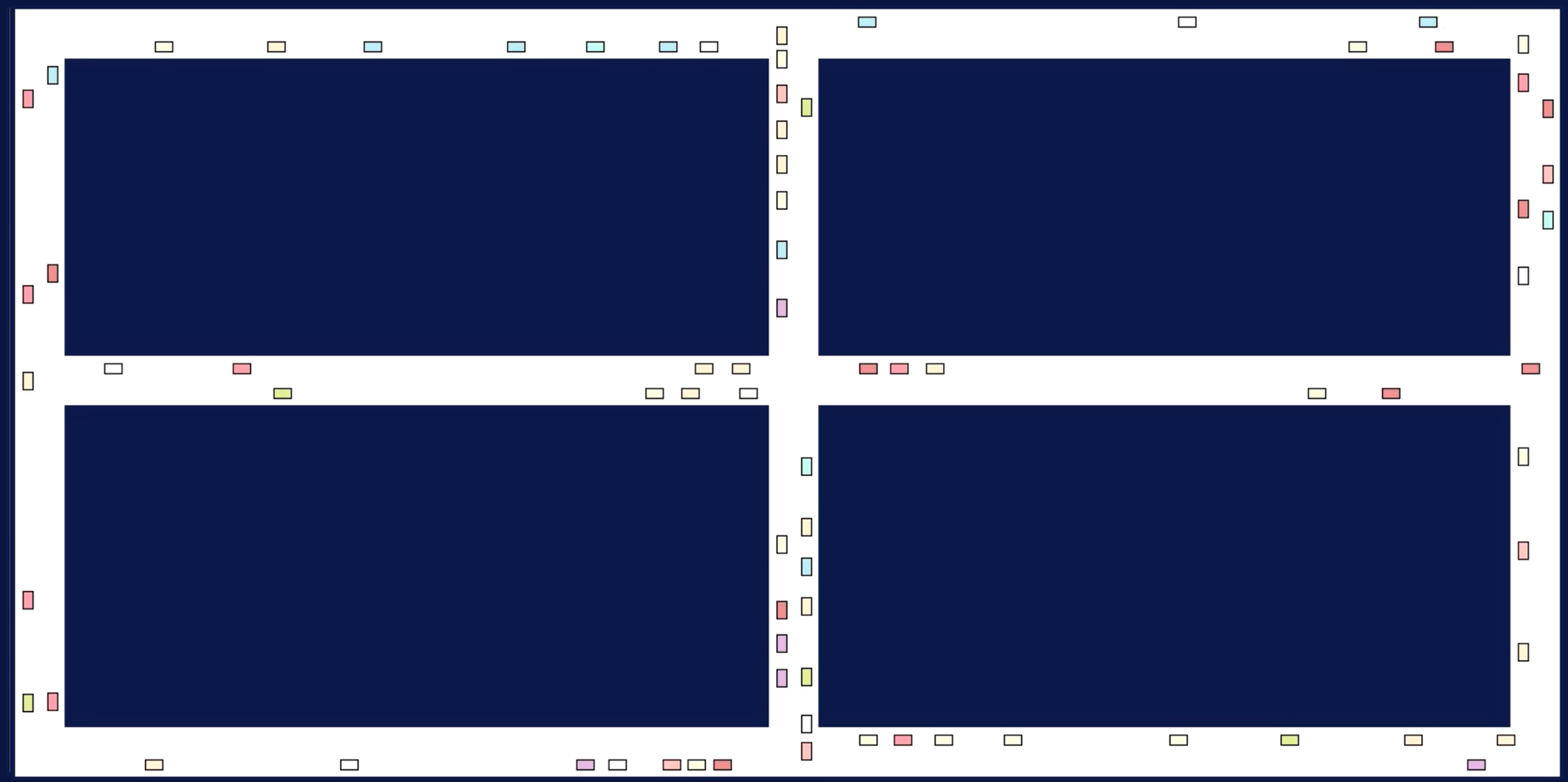


1 MODELISATION

2 SIMULATION / RESULTATS

3 PARADOXE DE BRAESS

SIMULATION POUR 80 VEHICULES

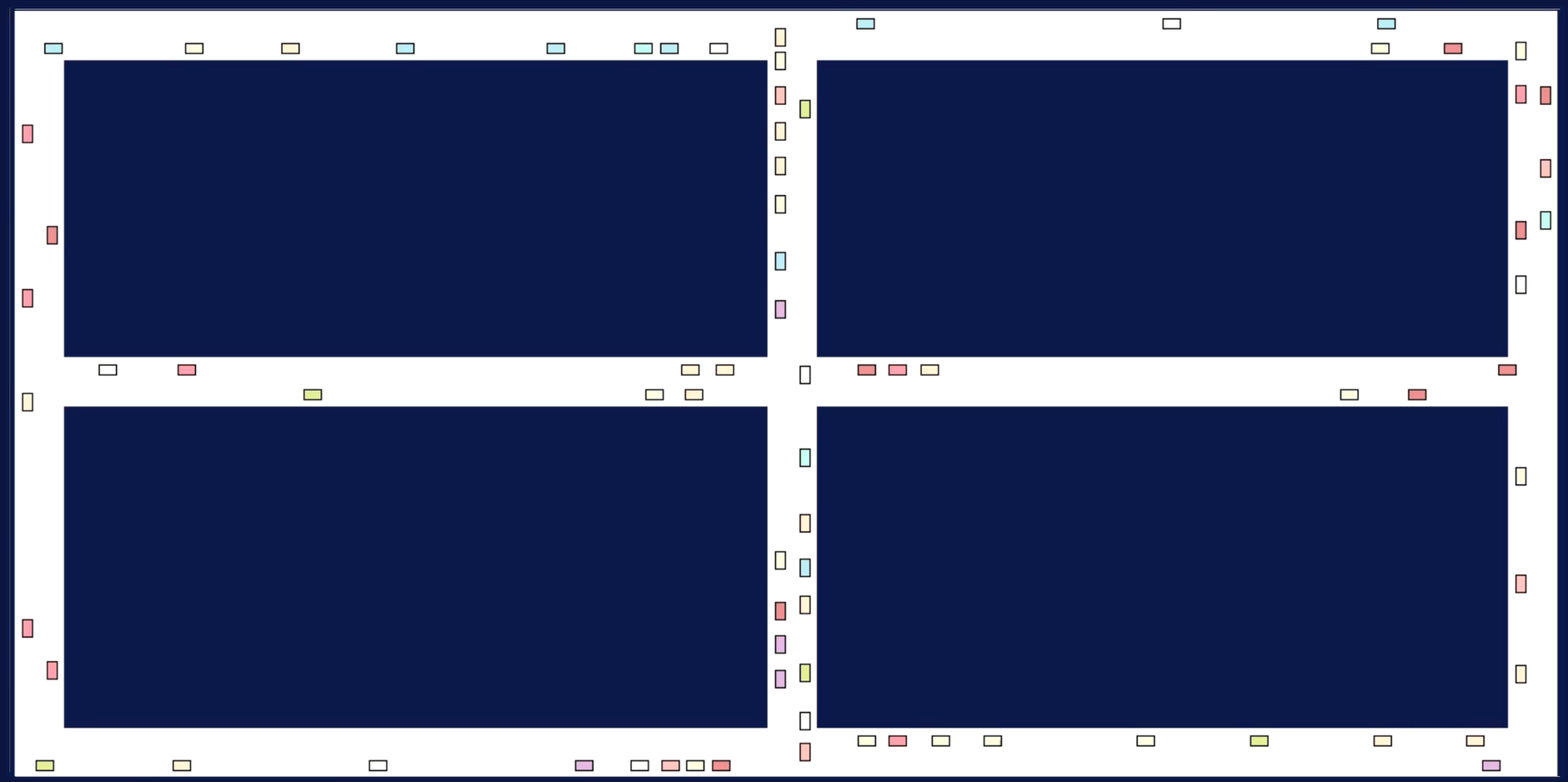


1 MODELISATION

2 SIMULATION / RESULTATS

3 PARADOXE DE BRAESS

SIMULATION POUR 80 VEHICULES

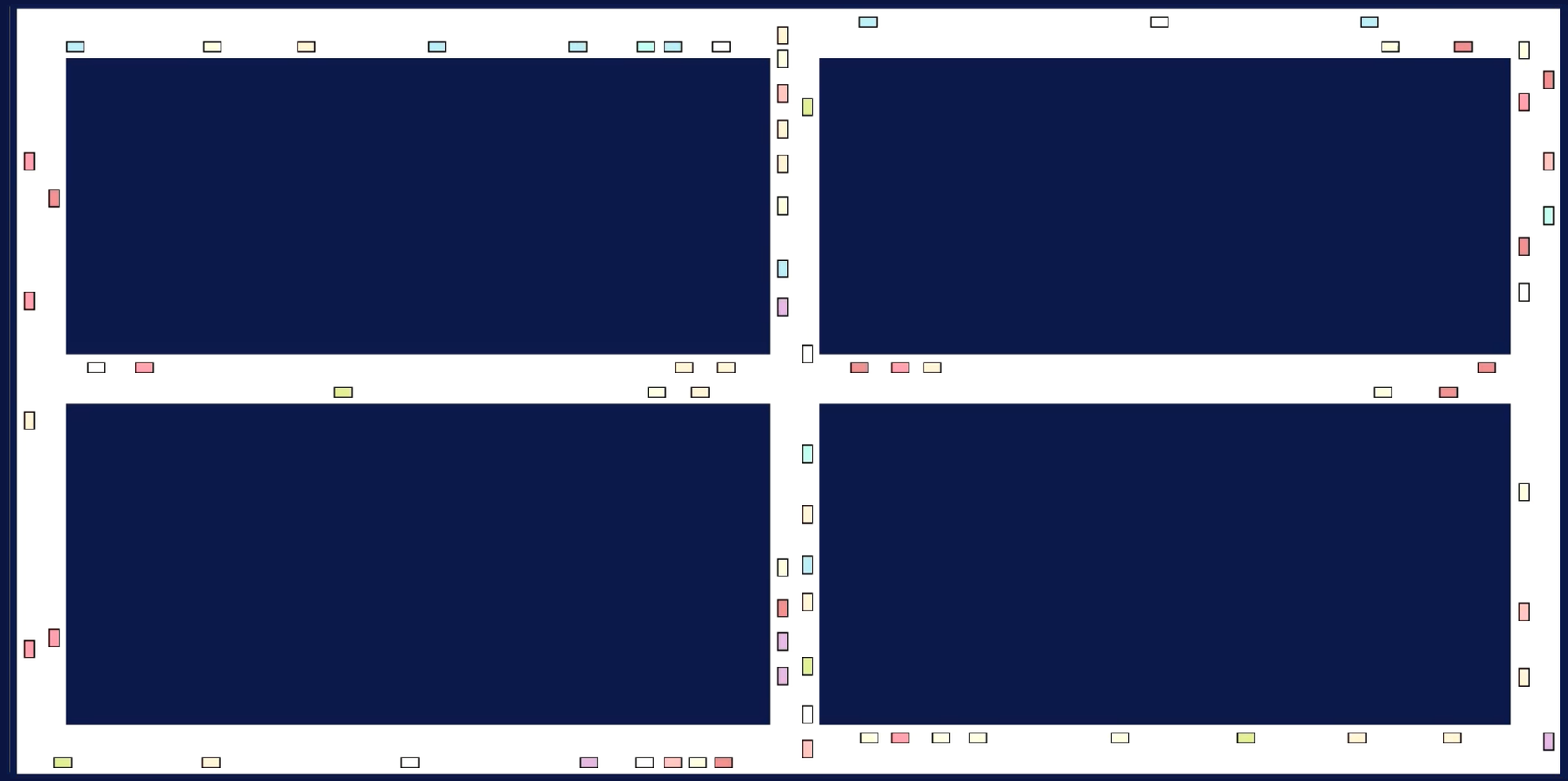


1 MODELISATION

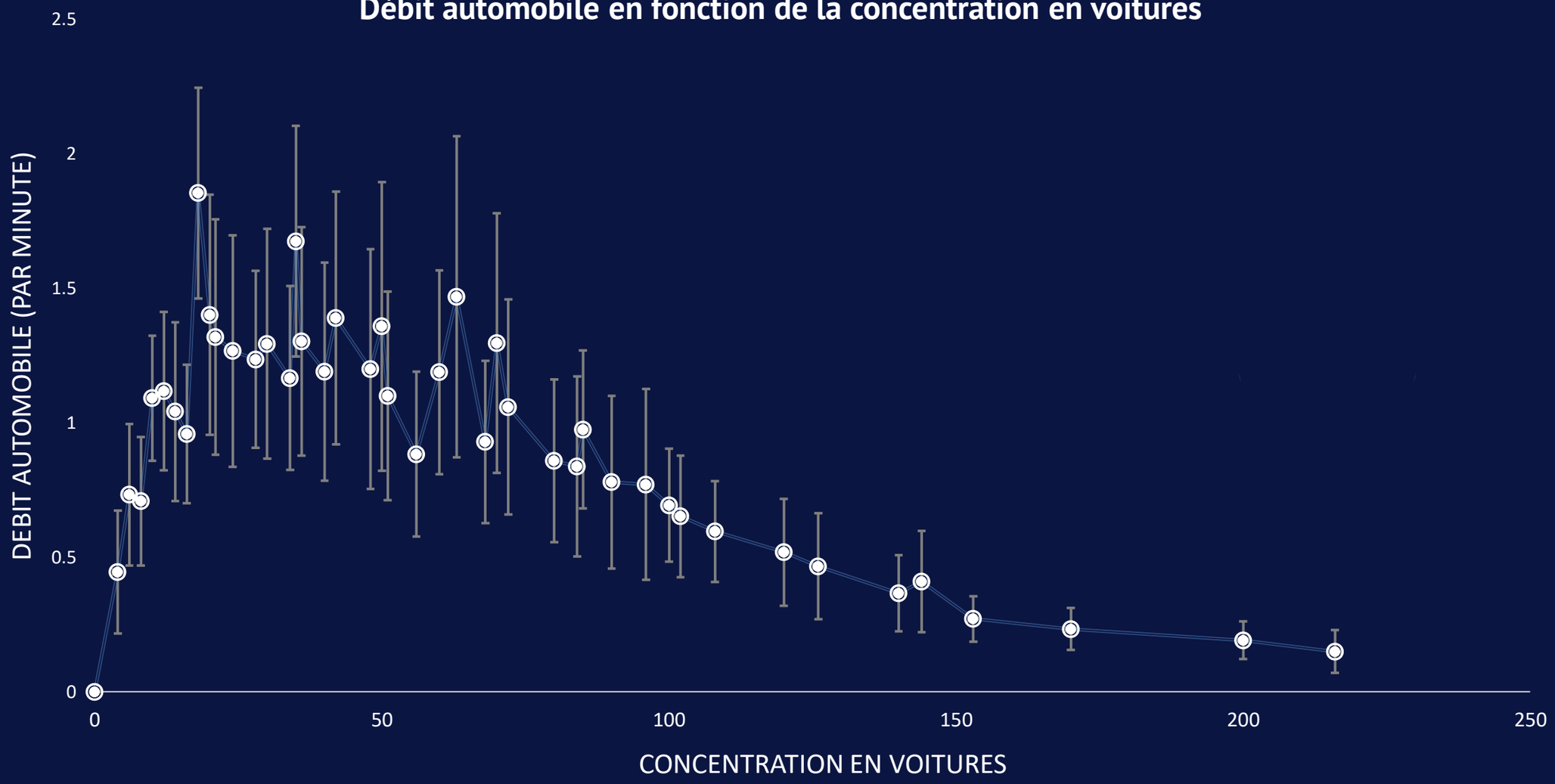
2 SIMULATION / RESULTATS

3 PARADOXE DE BRAESS

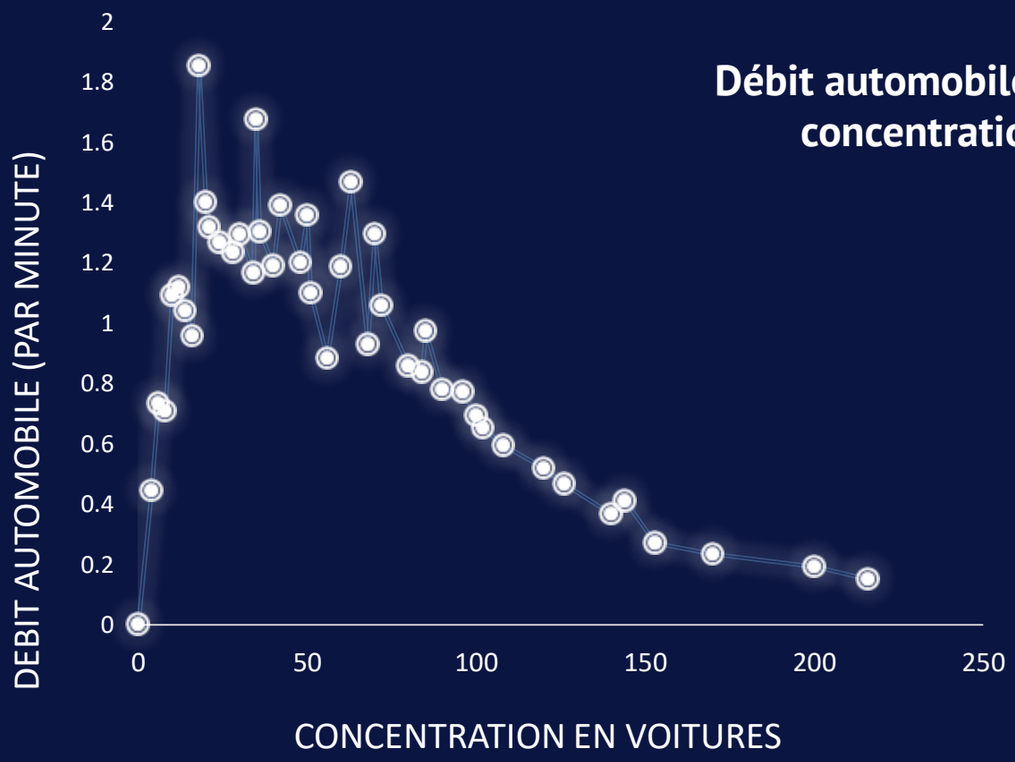
SIMULATION POUR 80 VEHICULES



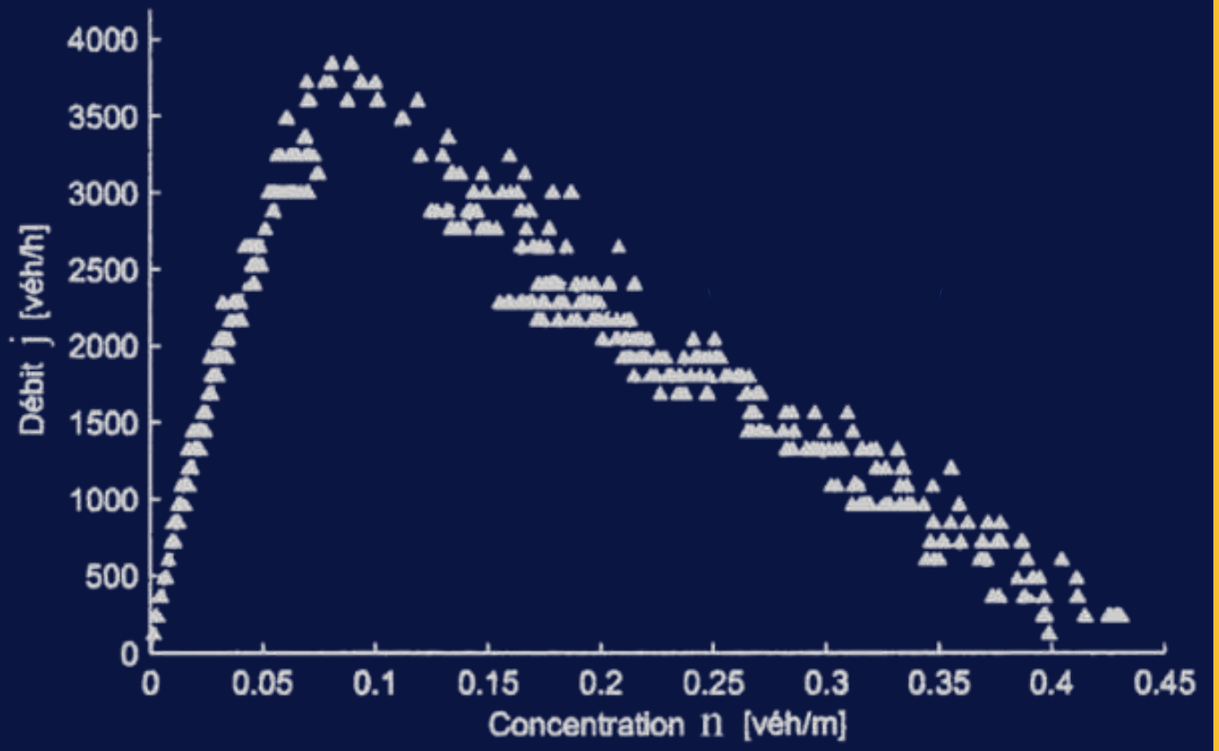
MESURES ET RESULTATS DE LA SIMULATION



COMPARAISON AUX RESULTATS EXPERIMENTAUX



Débit automobile en fonction de la concentration en voitures

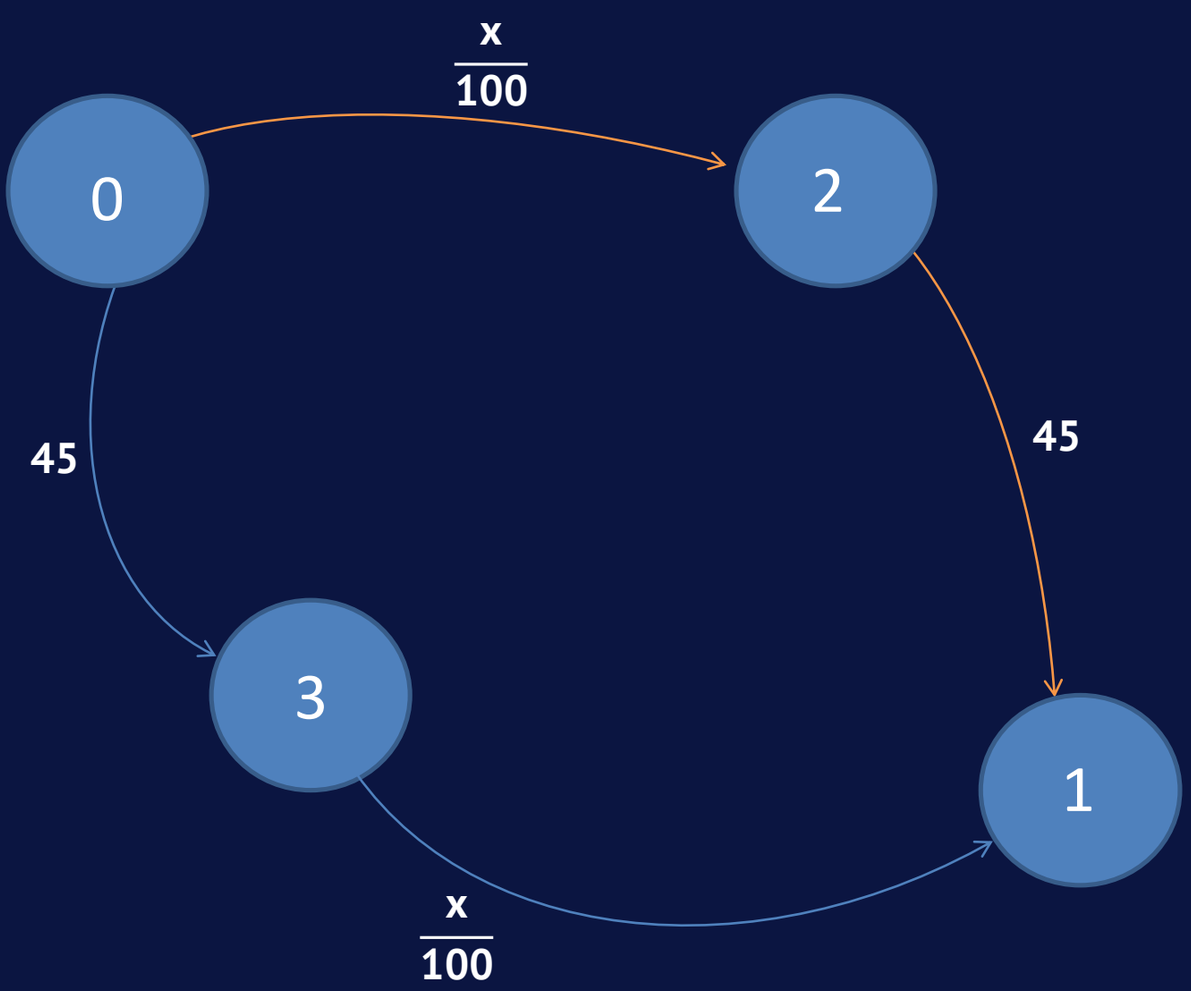


Source : Physique Centrale PSI 2005

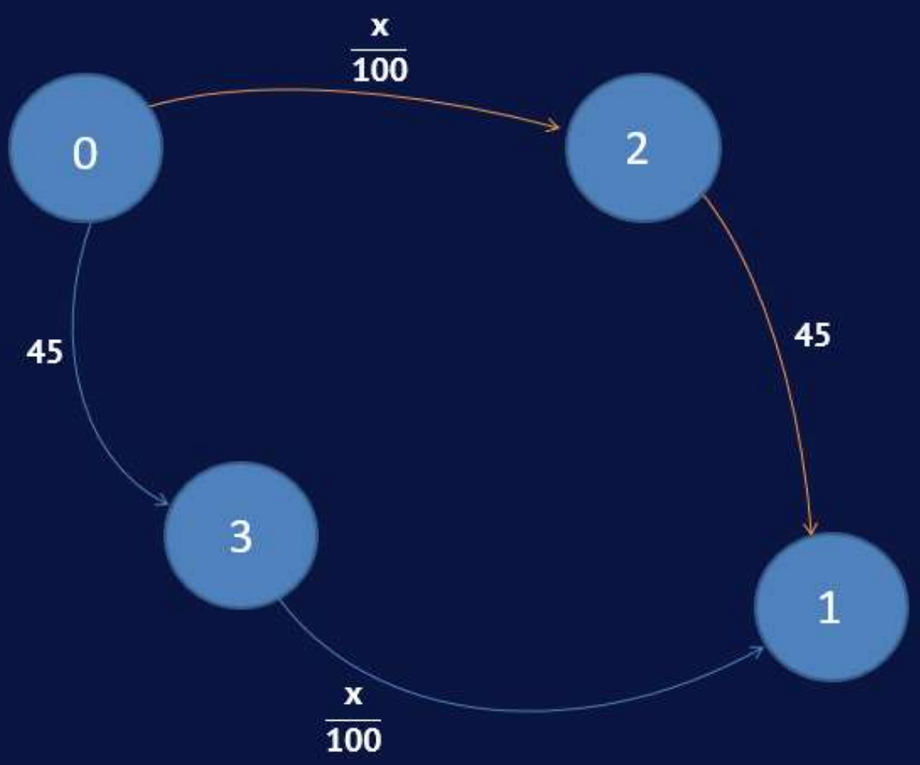
3 – PARADOXE DE BRAESS

1. MODELISATION
2. SIMULATION / RESULTATS
3. PARADOXE DE BRAESS

MODELISATION DU PHENOMENE



MODELISATION DU PHENOMENE



- **Temps social :**

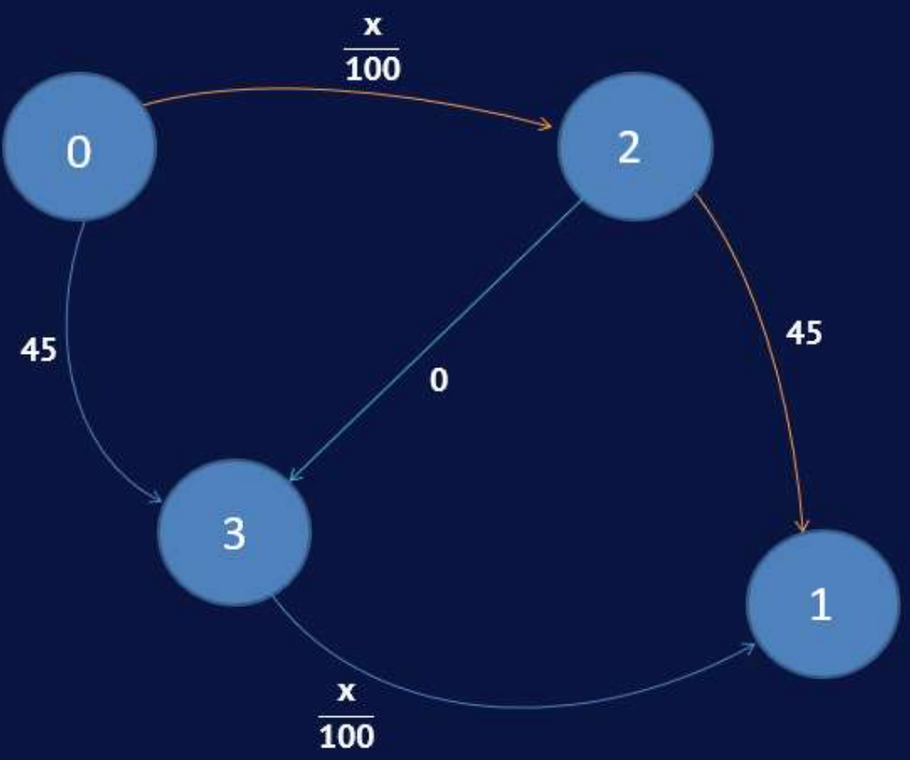
$$\frac{1}{2} \times \frac{x}{100} + 45 \text{ min}$$

- **Pour 4000 véhicules,**

$T_{social} = 65\text{min}$

1. MODELISATION
2. SIMULATION / RESULTATS
3. PARADOXE DE BRAESS

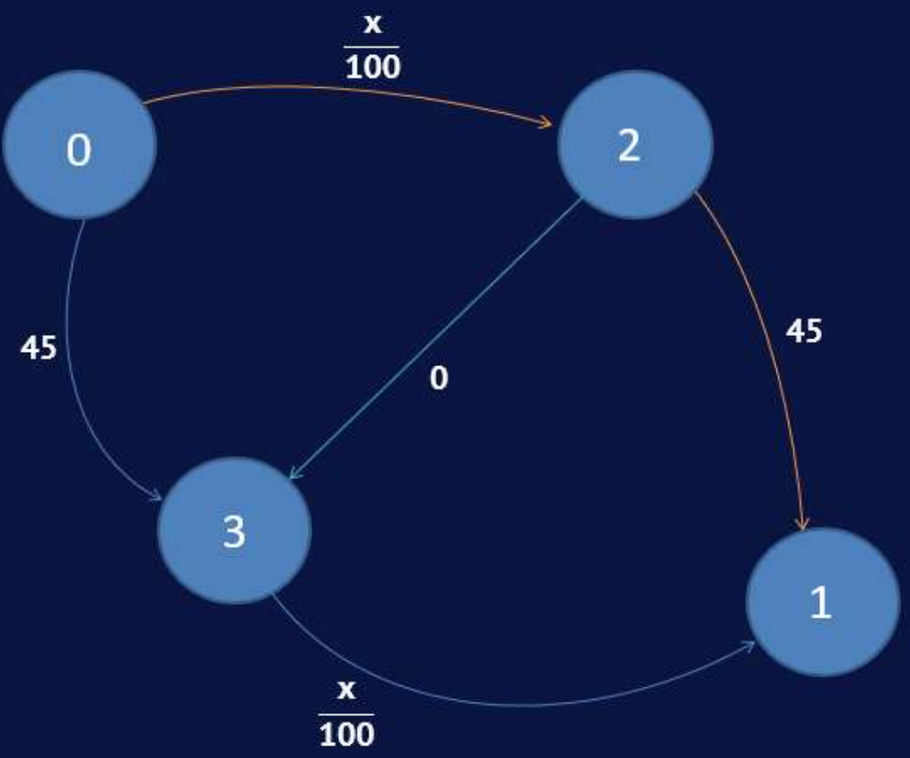
MODELISATION DU PHENOMENE



Apparition d'un 'raccourci' :

temps optimisé ?

MODELISATION DU PHENOMENE

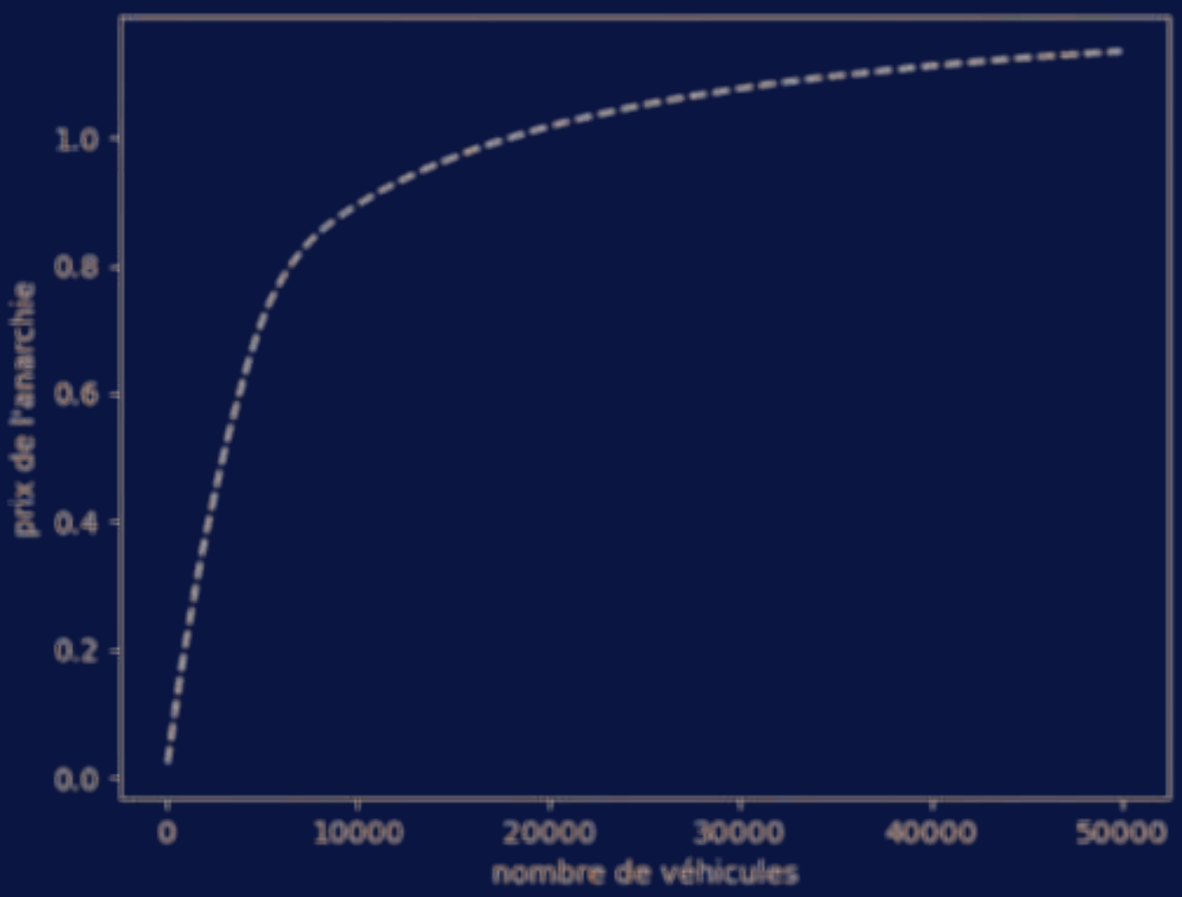


- **Plusieurs itinéraires possibles :**
  - **0-2-1**
  - **0-3-1**
  - **0-2-3-1 (chemin égoïste)**
- $T_{égoiste,initial} = 2 \times \frac{x}{100}$
- **Pour 4000 véhicules :** $T_{égoiste} = 80min$

Prix de l'anarchie = $\sup\{\frac{T_{égoiste}}{T_{social}}\}$

1. MODELISATION
2. SIMULATION / RESULTATS
3. PARADOXE DE BRAESS

SIMULATION NUMERIQUE



Asymptote horizontale en

+∞ en y = $\frac{4}{3} : \frac{T_{égoiste}}{T_{social}} \leq \frac{4}{3}$

ie prix de l'anarchie = $\frac{4}{3}$

ANNEXE

ANNEXE 1 : CONSTANTES PRÉDÉFINIES PAR L'IDM POUR UN VÉHICULE EN AGGLOMÉRATION

PARAMETRE	VALEUR PAR DEFAUT
a accélération maximale	1.0 $m.s^{-1}$
b décélération confortable	1.5$m.s^{-1}$
T délai de sécurité	1.0s
s_0 distance de sécurité minimale	2m
v_0 vitesse maximale	54 $km.h^{-1}$
δ exposant d'accélération	4

ANNEXE 2 : DEMONSTRATION DU PRIX DE L'ANARCHIE

- Supposons que $L_e(x)=aex+be$
- Energie de l'arête parcourue par x voitures :

$$E(e)=\sum_{i=1}^{x} L_e(i) = \sum_{k=1}^{x} a_e k+be = x\frac{a_e(x+1)+2be}{2}$$

- Temps passé par chaque conducteur sur e cumulé : $T(e)=xLe(x)$
- $E(e)\leqslant T(e)$
- $E(e)=x\frac{a_e(x+1)+2be}{2} \geqslant x\frac{a_e x+be}{2} = \frac{T(e)}{2}$

$$\rightarrow \frac{T(e)}{2} \leqslant E(e) \leqslant T(e)$$

ANNEXE 2 : DEMONSTRATION DU PRIX DE L'ANARCHIE

- Soit Z une distribution de voitures sur un graphe. Alors,

$$\frac{T_{social}(Z)}{2} \leqslant E(e) \leqslant Tsocial$$

- Soit Z une distribution socialement optimale, Z' une distribution 'égoïste' :

$$\begin{cases} \frac{T_{social}(Z')}{2} \leqslant E(Z') \\ E(Z') \leqslant E(Z) \\ E(Z) \leqslant Tsocial(Z) \end{cases}$$

$$\rightarrow T_{social}(Z') \leqslant 2Tsocial(Z)$$

ANNEXE 3 : CODE – SIMULATION DU PARADOXE DE BRAESS

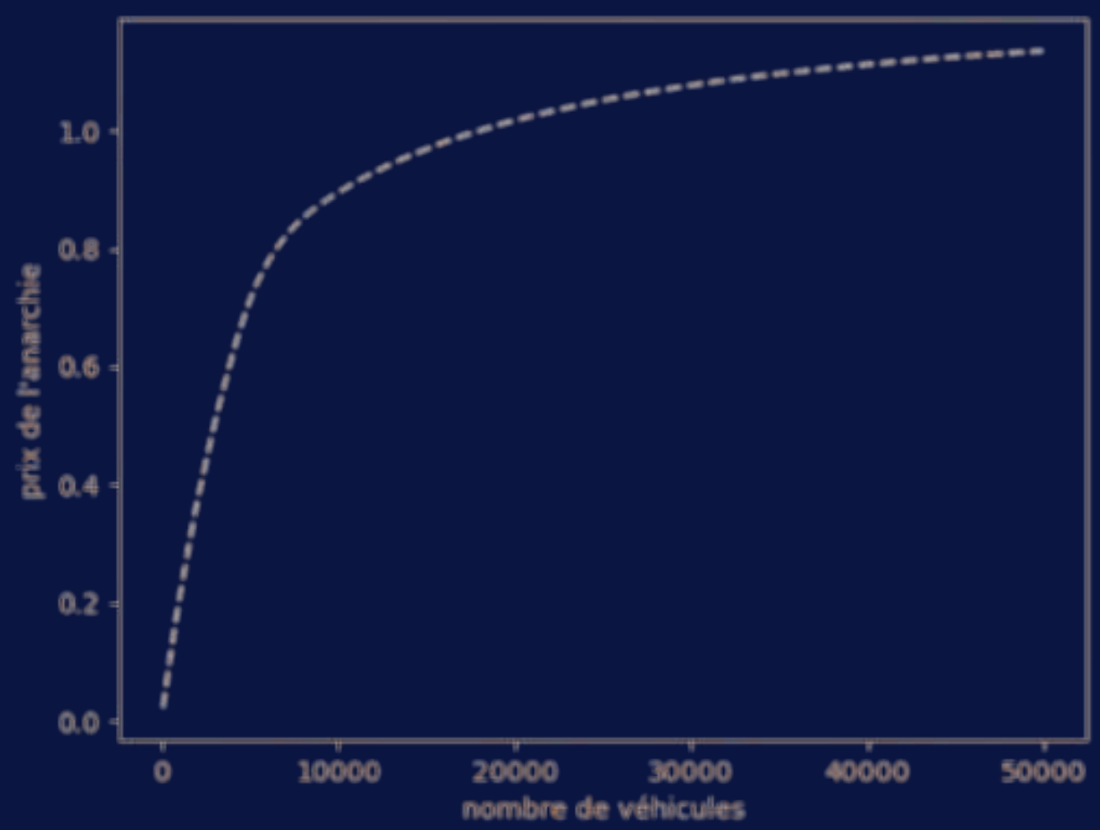
```
def anarchie (x):
    t_social = 2*(x/200 + 45)
    M = [[-1,x/100,45,-1],[-1,-1,0,45],[-1,-1,-1,x/100],[-1,-1,-1,-1]]
    chemin_ego = M[0][1] + M[1][2] + M[2][3]
    alter_A = M[0][1] + M[1][3]
    alter_B = M[0][2] + M[2][3]
    t = x*chemin_ego
    for i in range(x) :
        if chemin_ego<=alter_B and chemin_ego<=alter_A :
            return  t/x/t_social
        if alter_B<=min(alter_A,chemin_ego):
            M[0][1]-= 1/100
            chemin_ego = M[0][1] + M[1][2] + M[2][3]
            alter_A = M[0][1] + M[1][3]
            alter_B = M[0][2] + M[2][3]
            t+= alter_B - chemin_ego
        else :
            M[2][3] -=1/100
            chemin_ego = M[0][1] + M[1][2] + M[2][3]
            alter_A = M[0][1] + M[1][3]
            alter_B = M[0][2] + M[2][3]
            t+= alter_A - chemin_ego

    return (t/x/t_social)
```

ANNEXE 3 : CODE – SIMULATION DU PARADOXE DE BRAESS

```
def boucle (x) :
    nombre = []
    prix_anarchie = []
    for i in range(1,x) :
        nombre.append(100*i)
        prix_anarchie.append(anarchie(100*i))
    return (nombre,prix_anarchie)
```

```
a,b=boucle(200)
plt.figure()
plt.plot(a,b,'--')
plt.show()
```



ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
# importation des bibliotheques
import tkinter
import random
import numpy as np
from math import sqrt
import time

# creation de la fenetre et du canvas support a la
simulation
fenetre= tkinter.Tk()
fenetre_longueur = fenetre.winfo_screenheight()
fenetre_largeur = fenetre.winfo_screenwidth()
fenetre.title("TIPE-simulation")
fenetre.geometry(f'{fenetre_largeur}x{fenetre_longueur}')

canvas = tkinter.Canvas(fenetre)
canvas.configure(bg='#0B1541')
canvas.pack(fill="both", expand=True)
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
lotVoitures = [] #liste qui accueillira toutes les voitures

### variable utilisé pour la prise de mesure et l'écriture
dans le fichier excel correspondant
NbStat = 0
NbStatTab = []
NbVoitures = 4
nbTotalVoitures = 0
TempsStat = 10 #argument1
NbPointsStat = 5 #argument2

##
rafraichissement = 0.01#s #correspond à delta_t
(communémment "h") pour la methode d'Euler

# creation des classes Point, Voiture et Route
class Point :
  def __init__(self,x,y) :
    self.x = x
    self.y = y
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
class Route :
  connexions = []
  file_attente = [[],[],[],[],[],[],[],[],[]]
  matrice_adjacence = np.zeros((len(connexions), len(connexions)))
  intersections =
[[40,70],[1250,70],[40,350],[1250,350],[40,650],[1250,650],[650,70],[650,650],
[650,350]]
  proba_cumulee = np.zeros((len(connexions), len(connexions)))
  route_largeur = 20

  def __init__(self,i,j):
    self.i = i
    self.j = j
    self.construction_route()

  def construction_graphe() :
    for i in range (len(Route.intersections)) :
      Route.connexions.append(Point(Route.intersections[i][0],
Route.intersections[i][1]))
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
def construction_matrice() :
    Route.matrice_adjacence = np.zeros((len(Route.connexions),
len(Route.connexions)))
    Route.matrice_adjacence[0][2] = 0.5 # probabilité pour une voiture d'aller
de la route 0 à la route 2
    Route.matrice_adjacence[0][6] = 0.5
    Route.matrice_adjacence[1][3] = 0.5
    Route.matrice_adjacence[1][6] = 0.5
    Route.matrice_adjacence[2][0] = 0.5
    Route.matrice_adjacence[2][4] = 0.3
    Route.matrice_adjacence[2][8] = 0.2
    Route.matrice_adjacence[3][1] = 0.3
    Route.matrice_adjacence[3][5] = 0.4
    Route.matrice_adjacence[3][8] = 0.3
    Route.matrice_adjacence[4][2] = 0.5
    Route.matrice_adjacence[4][7] = 0.5
    Route.matrice_adjacence[5][3] = 0.5
    Route.matrice_adjacence[5][7] = 0.5
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
    Route.matrice_adjacence[6][0] = 0.4
    Route.matrice_adjacence[6][1] = 0.4
    Route.matrice_adjacence[6][8] = 0.2
    Route.matrice_adjacence[7][4] = 0.3
    Route.matrice_adjacence[7][5] = 0.3
    Route.matrice_adjacence[7][8] = 0.4
    Route.matrice_adjacence[8][2] = 0.2
    Route.matrice_adjacence[8][3] = 0.4
    Route.matrice_adjacence[8][6] = 0.2
    Route.matrice_adjacence[8][7] = 0.2

def proba_intersections ():
    Route.proba_cumulee = np.zeros((len(Route.connexions),
len(Route.connexions)))
    for i in range (len(Route.matrice_adjacence)) :
      Route.proba_cumulee[i] = np.cumsum(Route.matrice_adjacence[i])
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
    Route.matrice_adjacence[6][0] = 0.4
    Route.matrice_adjacence[6][1] = 0.4
    Route.matrice_adjacence[6][8] = 0.2
    Route.matrice_adjacence[7][4] = 0.3
    Route.matrice_adjacence[7][5] = 0.3
    Route.matrice_adjacence[7][8] = 0.4
    Route.matrice_adjacence[8][2] = 0.2
    Route.matrice_adjacence[8][3] = 0.4
    Route.matrice_adjacence[8][6] = 0.2
    Route.matrice_adjacence[8][7] = 0.2

def proba_intersections ():
    Route.proba_cumulee = np.zeros((len(Route.connexions),
len(Route.connexions)))
    for i in range (len(Route.matrice_adjacence)) :
      Route.proba_cumulee[i] = np.cumsum(Route.matrice_adjacence[i])
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
def construction_route(self):
    # la fonction permet simplement de tracer les routes sur le canvas
    canvas.create_rectangle(1250,50,1270,70, fill='#E3E0E0', width = 0)
    canvas.create_rectangle(20,50,40,70, fill='#E3E0E0', width = 0)
    canvas.create_rectangle(20,650,40,670, fill='#E3E0E0', width = 0)
    canvas.create_rectangle(1250,650,1270,670, fill='#E3E0E0', width = 0)
    if Route.matrice_adjacence[self.i][self.j] != 0 :
      if Route.connexions[self.i].x == Route.connexions[self.j].x :
        self.longueur_route = abs(Route.connexions[self.i].y-
Route.connexions[self.j].y)
        canvas.create_rectangle(Route.connexions[self.i].x-
Route.route_largeur,Route.connexions[self.i].y,Route.connexions[self.j].x+Rout
e.route_largeur,Route.connexions[self.j].y, fill='#E3E0E0',width=0)
      else :
        self.longueur_route = abs(Route.connexions[self.i].x-
Route.connexions[self.j].x)
        canvas.create_rectangle(Route.connexions[self.i].x,Route.connexions[se
lf.i].y-
Route.route_largeur,Route.connexions[self.j].x,Route.connexions[self.j].y+Rout
e.route_largeur, fill='#E3E0E0', width = 0)
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
class Voiture :
  vitesse_max = 30
  a = 1 # m/s^2 acceleration maximale
  b = 1.5 #m/s^2 deceleration confortable
  b_max = 10 #m/s^2 deceleration maximale
  T = 1 #s : delai de securite
  delta = 4 #: exposant d'acceleration
  distance_securite = 20 #m
  pos0=0
  def __init__(self, route_debut, route_fin, pas,dp) :
    self.route_debut = route_debut #route de depart de la voiture
    self.route_fin = route_fin #route de fin de la voiture
    self.dp = dp # vitesse de la voiture
    self.point_arrivees = pas # permet de connaitre l'avancement d'une voiture
sur sa route actuelle
    self.lead = self
    self.delta_v = self.lead.dp-self.dp #difference de velocite follower-
leader
    self.calcul_position()
    self.creation_voiture()
    self.acc = 0
    self.secu = 0
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
  def couleur():
    ## fonction colorant les véhicules (utile seuleement pour la représentation
graphique)
    liste =
['#d47d7d','#c4d080','#a1ced8','#c9a0c7','#eceadc','#a8e6cf','#dcedc1','#ffd3b6','#
ffaaa5','#ff8b94']
    return liste[random.randint(0,len(liste)-1)]

  def usine_a_voitures () :
    global NbVoitures
    global nbTotalVoitures
    for i in range(NbVoitures) :
      if i >= 9 :
        break
      for k in range (len(Route.matrice_adjacence[0])) :
        if Route.matrice_adjacence[i][k] != 0 :
          lotVoitures.append(Voiture(i,k,random.randint(0,250),1))
          lotVoitures.append(Voiture(k,i,random.randint(0,250),1))
          lotVoitures.append(Voiture(i,k,random.randint(0,250),1))
          lotVoitures.append(Voiture(k,i,random.randint(0,250),1))
          lotVoitures.append(Voiture(i,k,random.randint(0,250),1))

          nbTotalVoitures = nbTotalVoitures + 5
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
def calcul_position(self) :
    global NbStat
    # on calcule les coordonnées du debut et de la fin de la route sur laquelle se
trouve la voiture,
    # sa longueur, ainsi que le vecteur unitaire qui indique le sens et la direction
de deplacement
    # d'une voiture (du type [horizontale, verticale] )
    # exemple : [-1,0] -> la voiture se déplace selon les x décroissants
    self.point_depart = np.array([Route.connexions[self.route_debut].x,
Route.connexions[self.route_debut].y])
    self.point_arrivee = np.array([Route.connexions[self.route_fin].x,
Route.connexions[self.route_fin].y])
    self.u = (self.point_arrivee-
self.point_depart)/np.linalg.norm(self.point_arrivee-self.point_depart)

    pasConstante = 10
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
if self.u[0] == 0 and self.u[1] == -1 :
      self.point_depart = np.array([Route.connexions[self.route_debut].x+pasConstante,
Route.connexions[self.route_debut].y])
      self.point_arrivee = np.array([Route.connexions[self.route_fin].x+pasConstante,
Route.connexions[self.route_fin].y])
      self.longueur_r = abs(self.point_depart-self.point_arrivee)[1]
    elif self.u[0] == 0 and self.u[1] == 1 :
      self.point_depart = np.array([Route.connexions[self.route_debut].x-pasConstante,
Route.connexions[self.route_debut].y])
      self.point_arrivee = np.array([Route.connexions[self.route_fin].x-pasConstante,
Route.connexions[self.route_fin].y])
      self.longueur_r = abs(self.point_depart-self.point_arrivee)[1]
    elif self.u[0] == 1 and self.u[1] == 0 :
      self.point_depart = np.array([Route.connexions[self.route_debut].x,
Route.connexions[self.route_debut].y+pasConstante])
      self.point_arrivee = np.array([Route.connexions[self.route_fin].x,
Route.connexions[self.route_fin].y+pasConstante])
      self.longueur_r = abs(self.point_depart-self.point_arrivee)[0]
    elif self.u[0] == -1 and self.u[1] == 0 :
      self.point_depart = np.array([Route.connexions[self.route_debut].x,
Route.connexions[self.route_debut].y-pasConstante])
      self.point_arrivee = np.array([Route.connexions[self.route_fin].x,
Route.connexions[self.route_fin].y-pasConstante])
      self.longueur_r = abs(self.point_depart-self.point_arrivee)[0]
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
# tout ça dans le but de calculer les coordonnées x y de la position actuelle de la
voiture
    self.pos = self.point_depart + self.point_arrivees * self.u
    if self.route_debut == 0 and self.pos[0] != self.pos0 and round(self.pos[0],0)
== 300 :
      self.pos0 = self.pos[0]
      NbStat = NbStat + 1

  def creation_voiture(self) :
    self.apparence_voiture()
    self.rect = canvas.create_rectangle(self.pos[0]-self.largeur,
                    self.pos[1]-self.longueur,
                    self.pos[0]+self.largeur,
                    self.pos[1]+self.longueur,
                    fill = Voiture.couleur(), outline="Black", width=1)
  def apparence_voiture(self):
      if self.u[0] == 0 :
            self.largeur = 4
            self.longueur = 7
      else :
        self.largeur = 7
        self.longueur = 4
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
  def leader(self) :
    # on determine ici le 'leader' de chaque voiture
    maxi = self
    for voit in lotVoitures :
      if self.route_debut == voit.route_debut and self.route_fin == voit.route_fin
and maxi.point_arrivees <= voit.point_arrivees and self != voit :
        maxi = voit
    for voit in lotVoitures :
      if self.route_debut == voit.route_debut and self.route_fin ==
voit.route_fin and voit.point_arrivees >= self.point_arrivees and self != voit :
       if(voit.point_arrivees <= maxi.point_arrivees ) :
        maxi = voit
    self.lead = maxi
  def distance_secu (self) :
    s = Voiture.distance_securite + max(0,self.dp*Voiture.T
+(self.dp*self.delta_v)/(2*sqrt(Voiture.a*Voiture.b)))
    self.secu = s
    #print ('self.secu=',s)
    self.delta_v = self.dp-self.lead.dp
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
  def acceleration(self) :
    if self.lead == self and self.dp < Voiture.vitesse_max and self not in
file_croisement(lotVoitures)[self.route_fin]:
      ac = Voiture.a*(1-(self.dp/Voiture.vitesse_max)**Voiture.delta-
((self.secu)/(100000))**2)
      self.acc = ac
    elif self not in file_croisement(lotVoitures)[self.route_fin]:
      delta = self.lead.point_arrivees- self.point_arrivees-self.longueur*2
      if delta == 0 :
        delta = 0.0001
      ac = Voiture.a*(1-(self.dp/Voiture.vitesse_max)**Voiture.delta-
((self.secu)/delta)**2)
      self.acc = ac
    if self in file_croisement(lotVoitures)[self.route_fin]:
      self.acc = 0
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
  def changement_croisement(self) :
    ### si la voiture arrive a la fin de la route, on change de direction selon la
matrice de proba cumulee
    if abs(self.pos[0]-self.point_arrivee[0])<= 30 and abs(self.pos[1]-
self.point_arrivee[1]) <=30 :
      distribution = Route.proba_cumulee[self.route_fin]
      choix = self.route_debut
      while self.route_debut == choix :
        random_nombre = np.random.uniform()
        for i in range (len(distribution)) :
          if random_nombre <= distribution[i] :
            choix = i
            break
        if random_nombre > distribution[-1] :
          choix = np.argmax(distribution > random_nombre)
      self.route_debut = self.route_fin
      self.route_fin = choix
      self.point_arrivees = 0
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
def mise_a_jour(self) :
  #mise à jour des états des véhicules (vitesse, position)
  self.delta_v = self.dp-self.lead.dp
  if self.dp + self.acc < 0 :
    self.dp = 0
    self.point_arrivees += (1/2*self.dp**2)/self.acc
  else :
    self.dp += self.acc*rafraichissement
    self.point_arrivees += self.dp + 1/2*self.acc*rafraichissement**2

def freinage(self) :
  #freinage random d'apres NaSch
  rando = random.randint(1,100)
  if rando<20 :
    self.dp -=self.dp/2
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
def file_croisement(voitures) :
  pas_croisement=60
  for v in voitures :
    if v.longueur_r-v.point_arrivees < pas_croisement and v not in
Route.file_attente[v.route_fin] :
      Route.file_attente[v.route_fin].append(v)
  return Route.file_attente

def gestion_croisement(v) :
  for i in Route.file_attente :
    if len(i)>=1 :
      i[0].dp = 0.75
  for i in Route.file_attente :
    if v in i :
      if v == i[0] :
        if v.point_arrivees >= 10 and v.point_arrivees <30:
          v.dp = 1
          i.remove(v)
      else:
        v.dp = 0
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
def animate () :
  file_croisement(lotVoitures)
  for v in lotVoitures :
    v.mise_a_jour()
    v.leader()
    v.distance_secu()
    v.acceleration()
    v.freinage()
    gestion_croisement(v)
    v.changement_croisement()
    v.apparence_voiture()
    v.calcul_position()

def affichage() :
  for v in lotVoitures :
    canvas.coords(v.rect, v.pos[0]-v.largeur, v.pos[1]-
v.longueur,v.pos[0]+v.largeur,v.pos[1]+v.longueur)
    fenetre.update()
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
def init():
  global NbVoitures
  global TempsStat
  global NbPointsStat
  Route.construction_graphe()
  Route.construction_matrice()
  Route.proba_intersections()
n = len(Route.matrice_adjacence)
for i in range (n) :
  for j in range(i+1,n):
    Route(i,j)
if len(sys.argv) > 1 :
  Voiture.vitesse_max = int(sys.argv[1])
if len(sys.argv) > 2 :
  TempsStat = int(sys.argv[2])
if len(sys.argv) > 3 :
  NbPointsStat = int(sys.argv[3])
Voiture.usine_a_voitures()
```

ANNEXE 4 : CODE – SIMULATION DU TRAFIC

```
init()
print ('vitesse_max=',Voiture.vitesse_max)
print ('TempsStat=',TempsStat)
print ('NbPointsStat=',NbPointsStat)
strPrint = str(nbTotalVoitures)
strPrint = strPrint + ";"
f = open('C:\majda\cpge-mp\\statVoitures2.csv', 'a')
f.write(str(nbTotalVoitures))
f.write(";")
t = time.time()
t1 = time.time()
while True :
  delai = time.time()- t
  delaiTotal=time.time()- t1
  if delai > 0.02:
    animate()
    affichage()
    t = time.time()
```

ANNEXE 5 : CODE – SIMULATIONS SIMULTANEES

```
set /a vitesse_max=50

set /a countVitesse_max = 0

:boucleVoitures
set /a countVitesse_max = countVitesse_max + 2
if %countVitesse_max%==%vitesse_max% goto
finboucleVoitures
echo suite_%countVitesse_max%
py tipe60.py %countVitesse_max% 60 9
goto boucleVoitures

:finboucleVoitures
echo fin boucles Voitures
```

ANNEXE 5 : CODE – SIMULATIONS SIMULTANEES

```
set /a nbInstances=5

set /a countInstance = 0

:circuit
set /a countInstance = countInstance + 1
if %countInstance%==%nbInstances% goto
finBoucleInstance
echo suiteInstance_%countInstance%
start lanceInstance.bat
goto circuit

:finboucleInstance
echo fin de toutes les instances
```