

Modélisation du trafic routier : recherche de solutions au problème des embouteillages



Kettaf Riyad 40680
Viguier William 41880

Problème de la congestion routière

- 10 jours perdus dans les embouteillages par an, par personne
- 15 % des émissions de CO₂ du trafic routier
- Duplication des émissions lors des situations de congestion
- 22 milliards d'euros de pertes en France prévues en 2030

Sommaire

I-Modélisation de l'interaction élémentaire entre véhicules:
implémentation de l'IDM

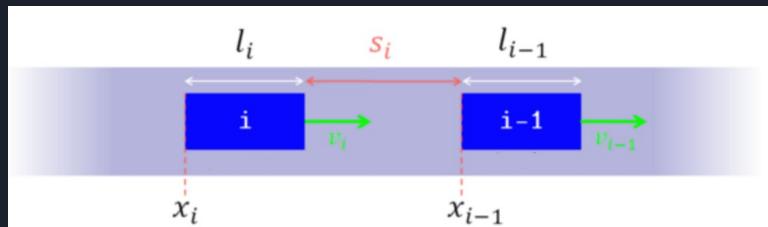
II-Mise en place d'un système routier à échelle mésoscopique

III-Étude du comportement usager

IV-Étude de solutions réalisables afin de limiter la congestion

IDM en interaction

$$\frac{dv_i}{dt} = a_{max} \left(1 - \left(\frac{v_i}{v_{0,i}} \right)^\delta - \left(\frac{S_{0,i} + v_i T_i + \frac{v_i \Delta v}{2\sqrt{a_{max} b}}}{S_i} \right)^2 \right)$$



- v_i : vitesse du véhicule i
- $v_{0,i}$: vitesse maximale souhaitée du véhicule i
- a_{max} : accélération maximale du véhicule i
- $S_{0,i}$: distance minimale souhaitée entre le véhicule i et i-1
- T_i : Temps de réaction du i-ème conducteur
- S_i : distance entre l'avant du véhicule i et i-1
- δ : Coefficient d'accélération (empirique)
- Δv : $v_{i-1} - v_i$
- b : décélération maximale

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

Paramètres de l'IDM

Source: Limitations and Improvements of the Intelligent Driver Model (IDM)

$$\frac{dv_i}{dt} = a_{max} \left(1 - \left(\frac{v_i}{v_{0,i}} \right)^\delta - \left(\frac{S_{0,i} + v_i T_i + \frac{v_i \Delta v}{2\sqrt{a_{max} b}}}{S_i} \right)^2 \right)$$

$$a_{max} = 0.73 \text{ m. s}^{-2}$$

$$b = 1.67 \text{ m. s}^{-2}$$

$$v_{0,i} = 30 \text{ m. s}^{-1}$$

$$T_i = 1.6 \text{ s}$$

$$S_{0,i} = 2 \text{ m}$$

$$\delta = 4$$

$-v_i$: vitesse du véhicule i

$-v_{0,i}$: vitesse maximale souhaitée du véhicule i

$-a_{max}$: accélération maximale du véhicule i

$-S_{0,i}$: distance minimale souhaitée entre le véhicule i et i-1

$-T_i$: Temps de réaction du i-eme conducteur

$-S_i$: distance entre l'avant du véhicule i et i-1

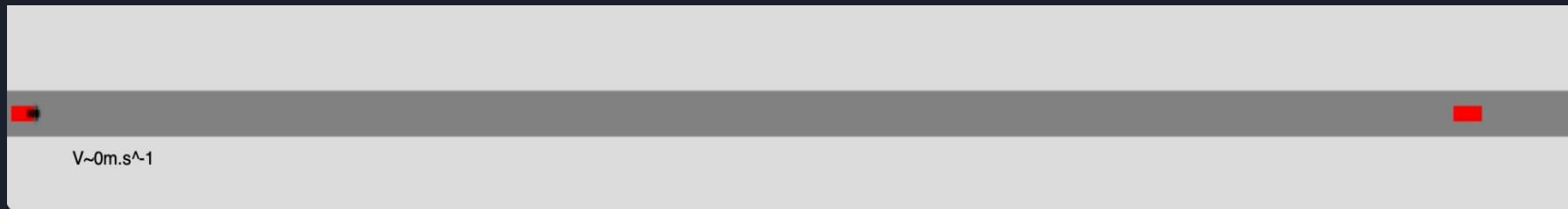
$-\delta$: Coefficient d'accélération (empirique)

$-\Delta v$: $v_{i-1} - v_i$

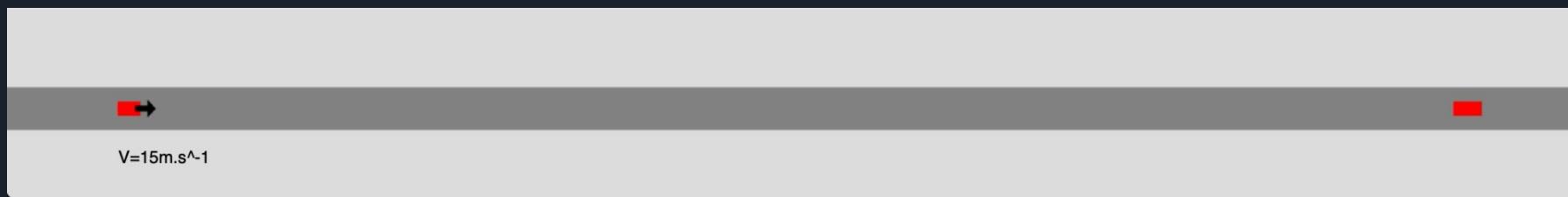
$-b$: décélération maximale

Cohérence du modèle

t=0s



t=10s

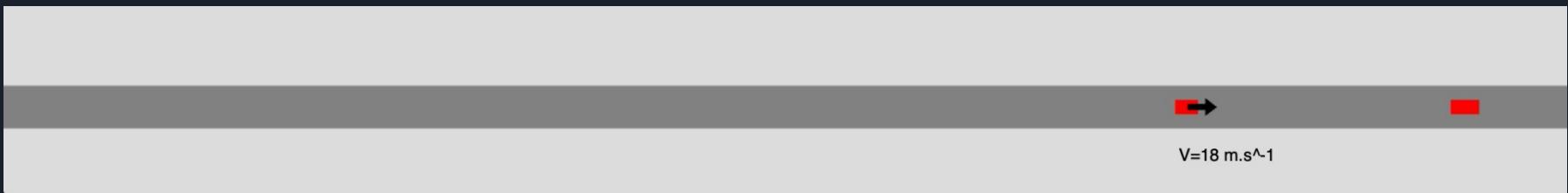


t=20s

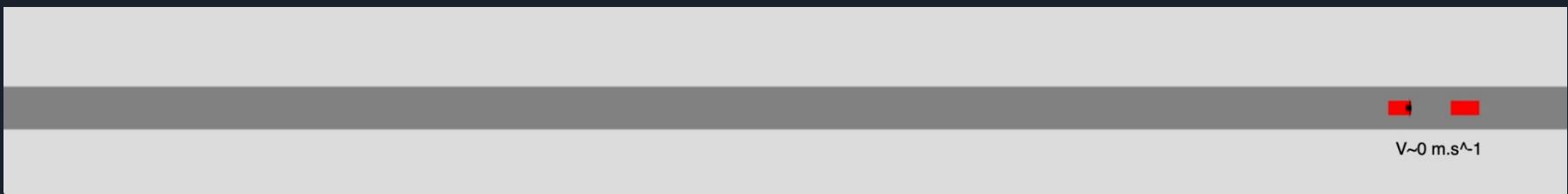


Cohérence du modèle

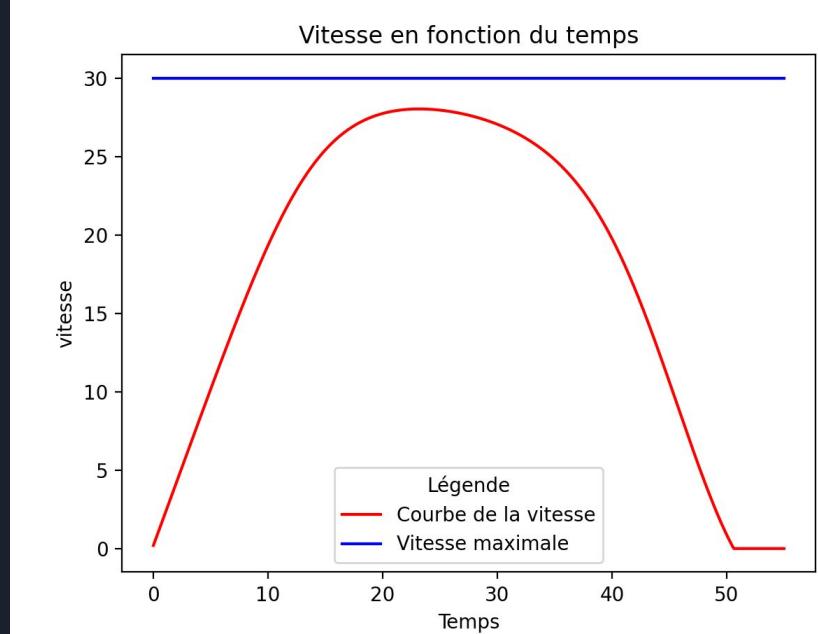
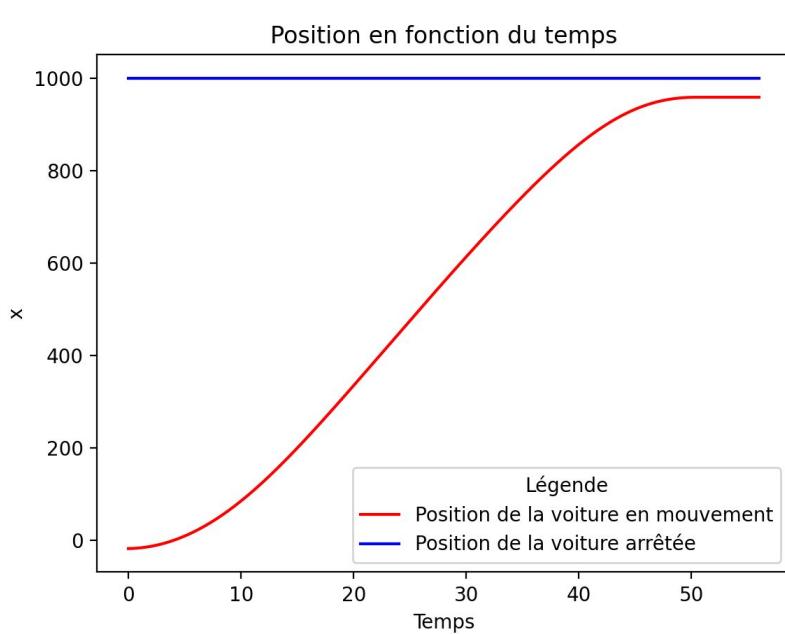
t=40s



$t \rightarrow +\infty$



Profils des grandeurs mécaniques

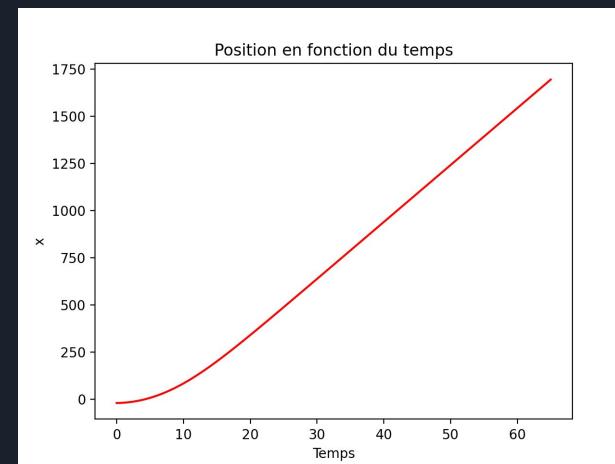
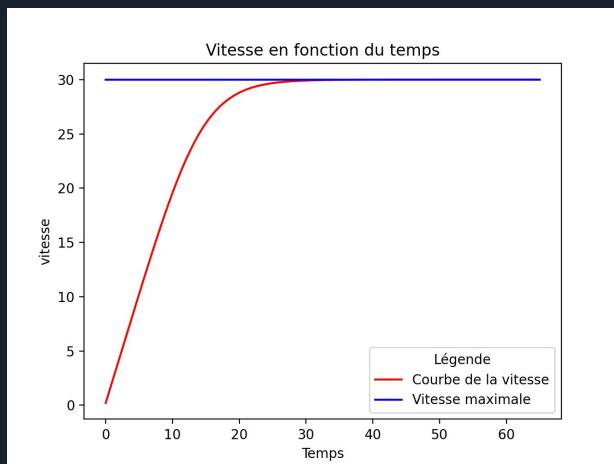


-Continuité

-Phases du mouvement

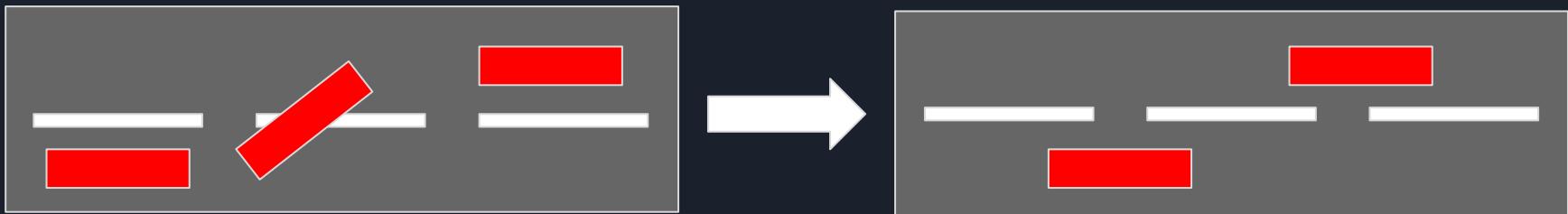
IDM sur voie libre

$$S_i \rightarrow +\infty \implies \frac{dv_i}{dt} = a_i \left(1 - \left(\frac{v_i}{v_{0,i}} \right)^\delta \right)$$

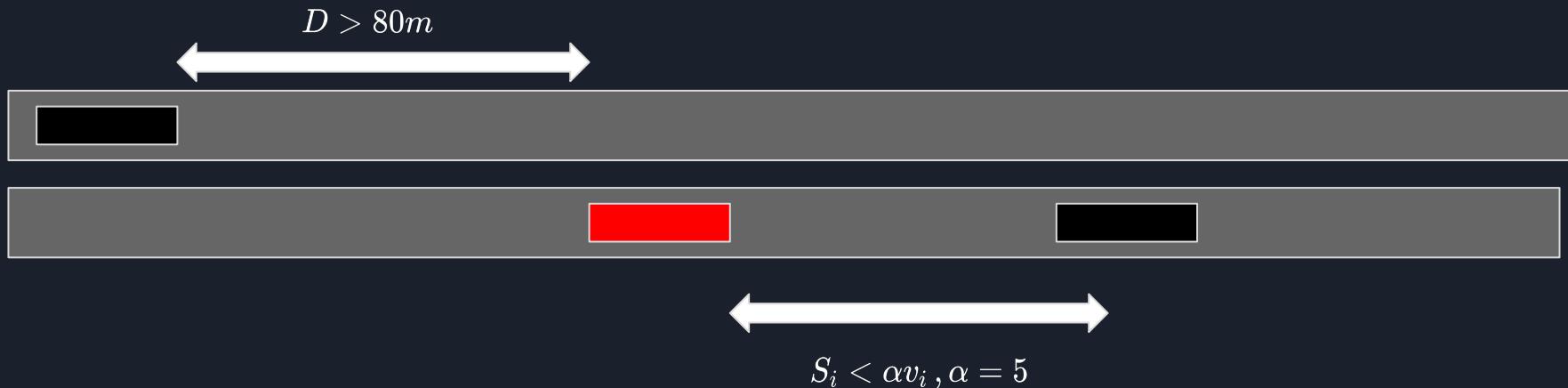


Voies multiples: changements de voies

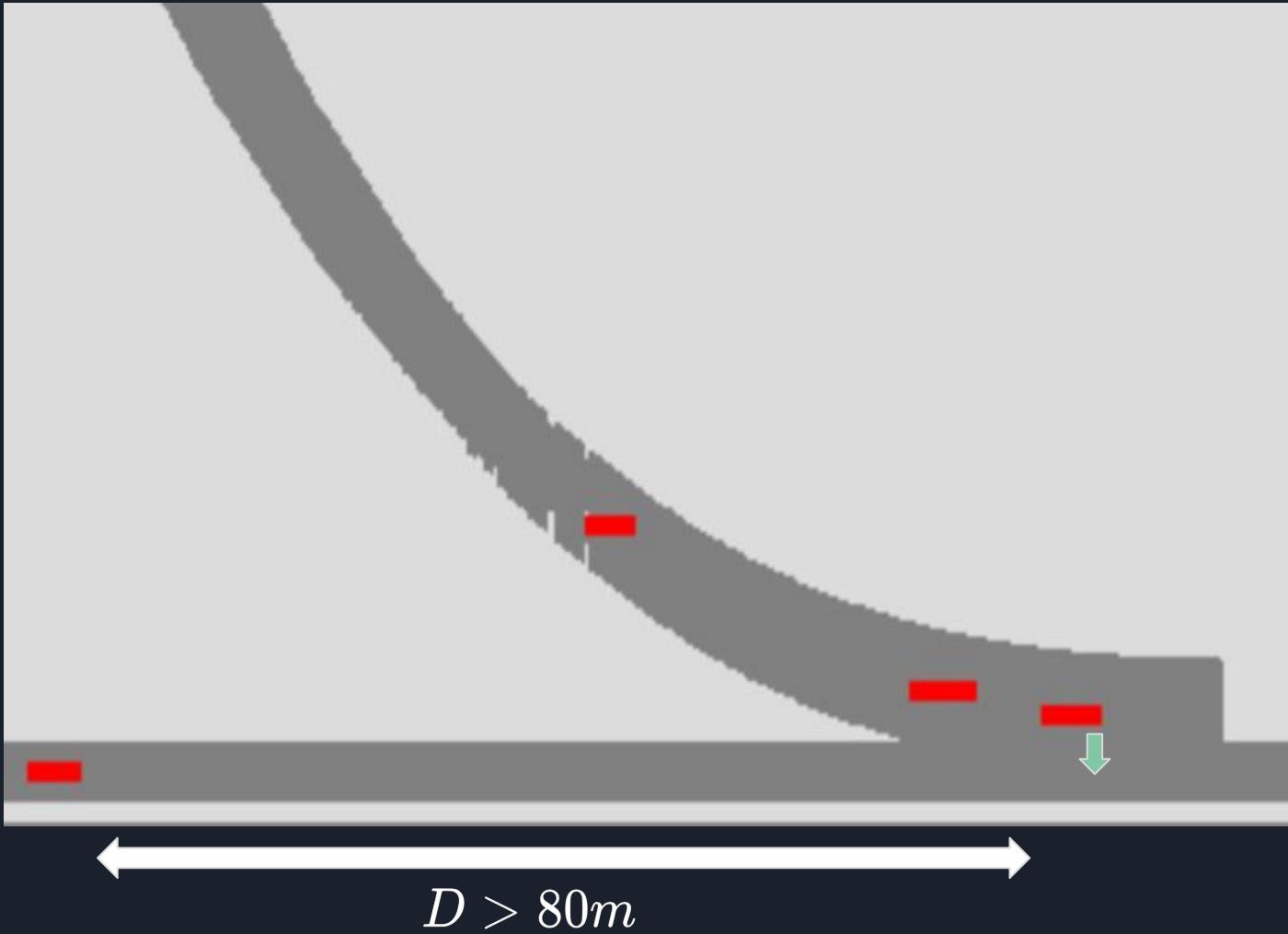
-Simplification de la réalité:



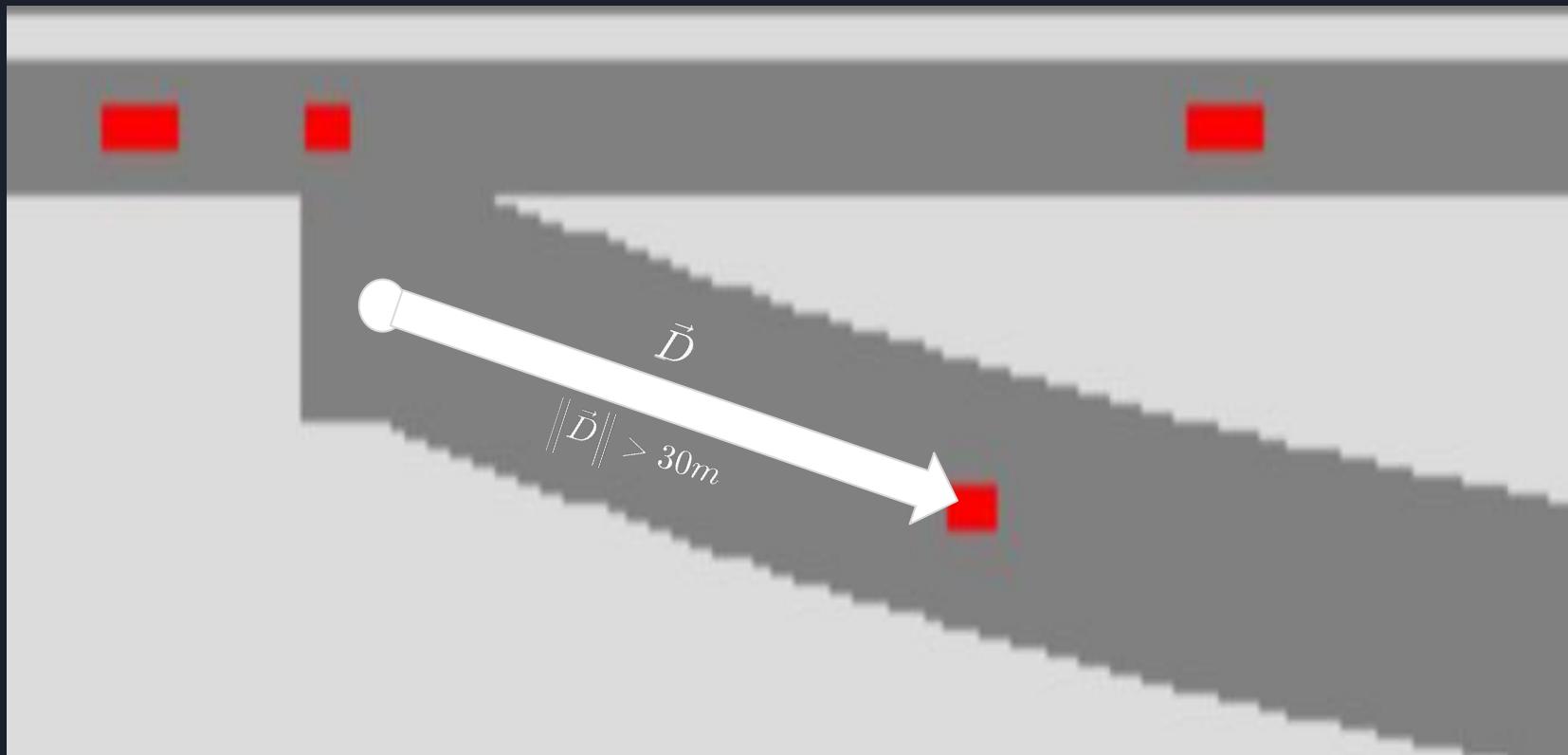
-Prise en compte des contraintes du code de la route:



Voies d'insertions

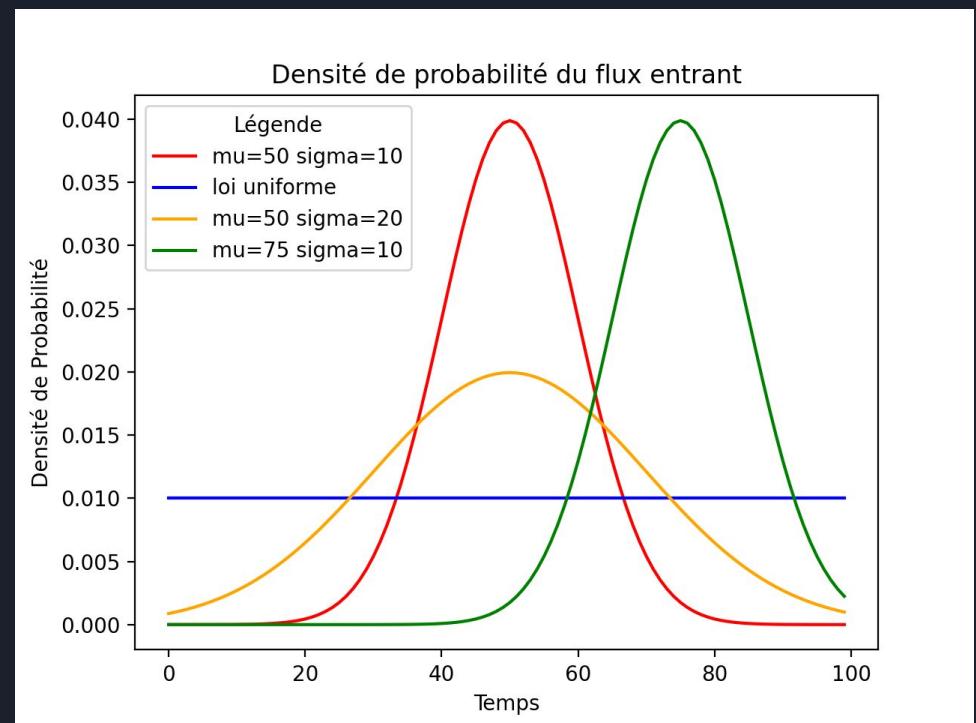


Voies de sorties



Calcul du flux entrant

$$P(\text{apparition}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2}$$



Variations des paramètres de l'IDM à l'échelle individuelle

$$a_{\max} = 0.73 \text{ m. s}^{-2}$$

$$b = 1.67 \text{ m. s}^{-2}$$

$$v_{0,i} = 30 \text{ m. s}^{-1}$$

$$T_i = 1.6 \text{ s}$$

$$S_{0,i} = 2 \text{ m}$$

$$\delta = 4$$

Distribution gaussiennes



$$a_{\max} = G(0.73, 0.2) \text{ m. s}^{-2}$$

$$b = G(1.67, 0.2) \text{ m. s}^{-2}$$

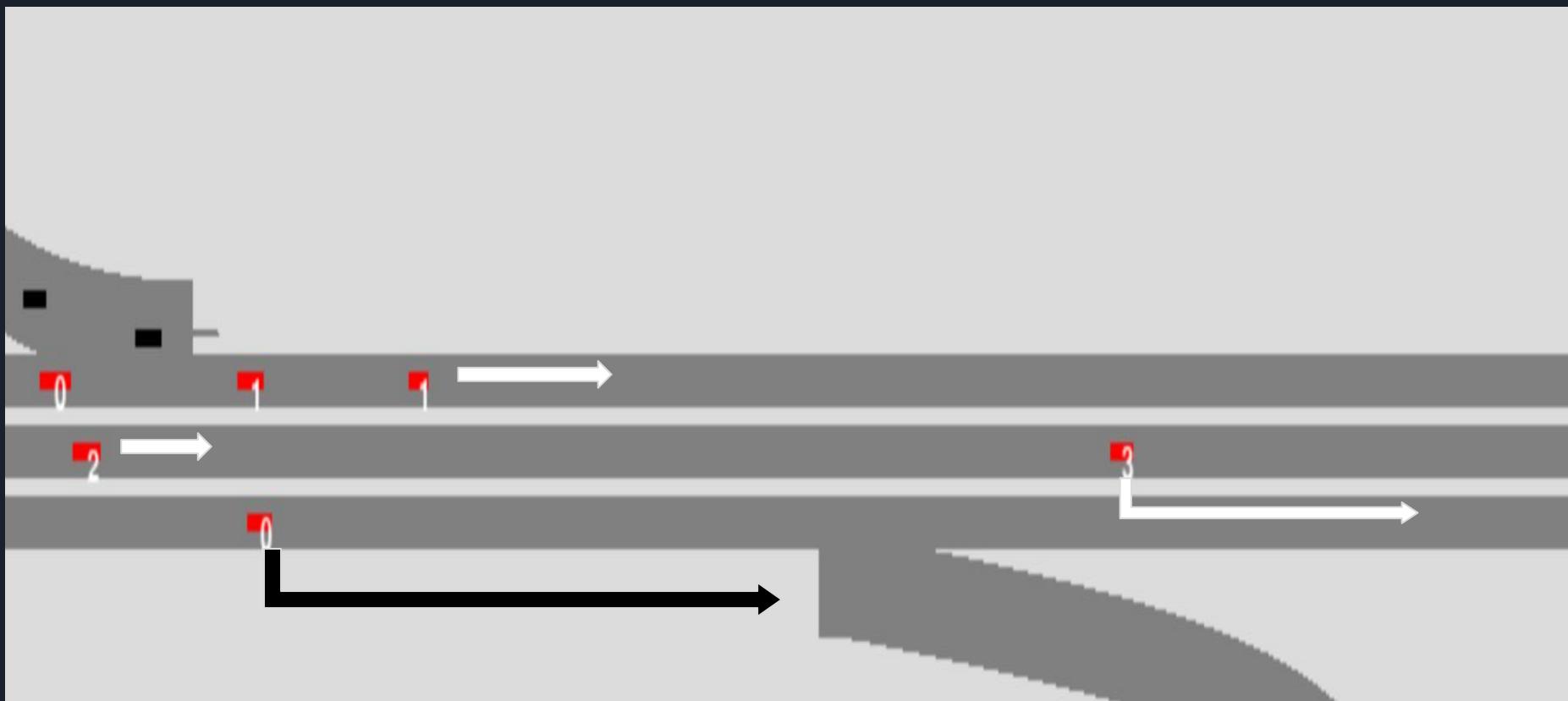
$$v_{0,i} = 30 \text{ m. s}^{-1}$$

$$T_i = G(1.6, 0.5) \text{ s}$$

$$S_{0,i} = 2 \text{ m}$$

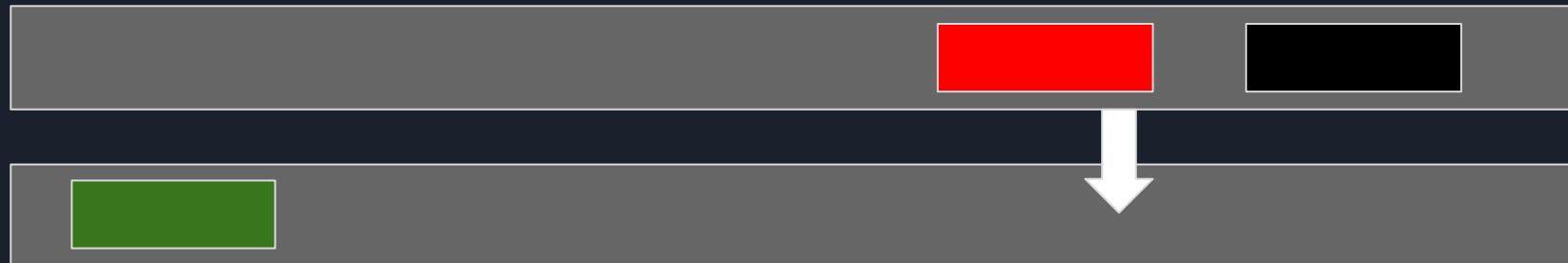
$$\delta = 4$$

Objectifs de parcours



Altruisme contre égoïsme

- Laisse les véhicules immobiles devant se rabattre



$$v \rightarrow 0 m \cdot s^{-1}$$

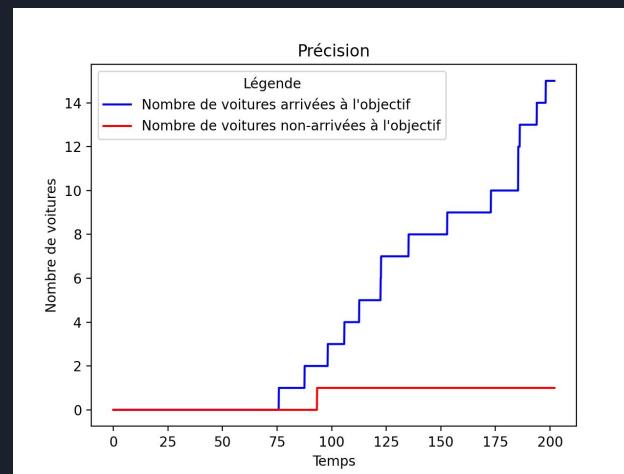
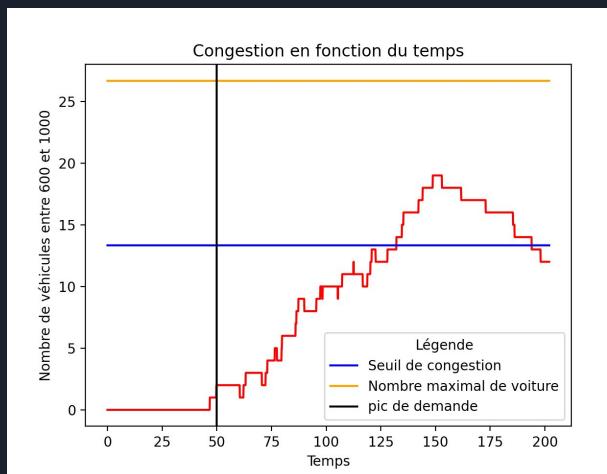
- Change de voie
 \iff Se rapproche de l'objectif et aucun véhicule ralenti sur la voie

Effets de l'égoïsme

$t = 100s, \sigma = 20$

$600 m < \text{Zone d'étude} < 1000 m$

$$\begin{aligned}\mu &= 50 \\ \sigma &= 20\end{aligned}$$



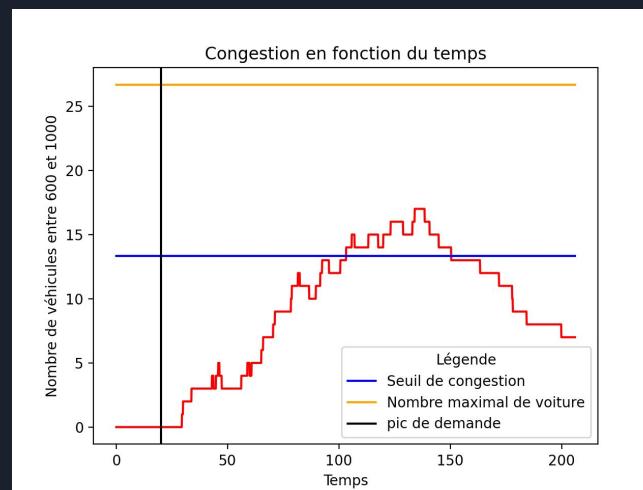
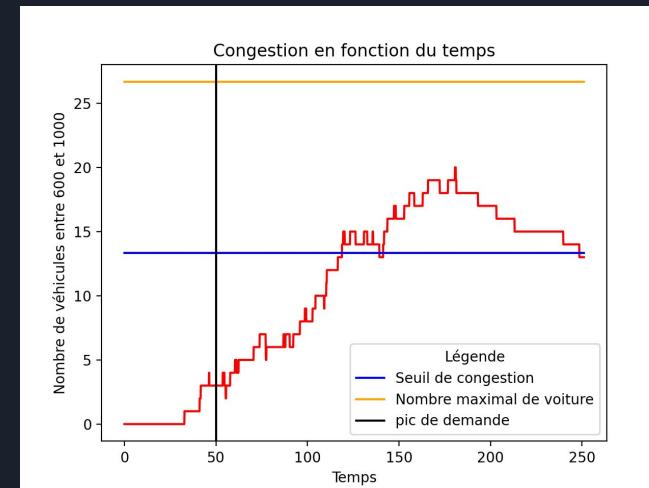
Effets de l'égoïsme

$$\mu = 50$$

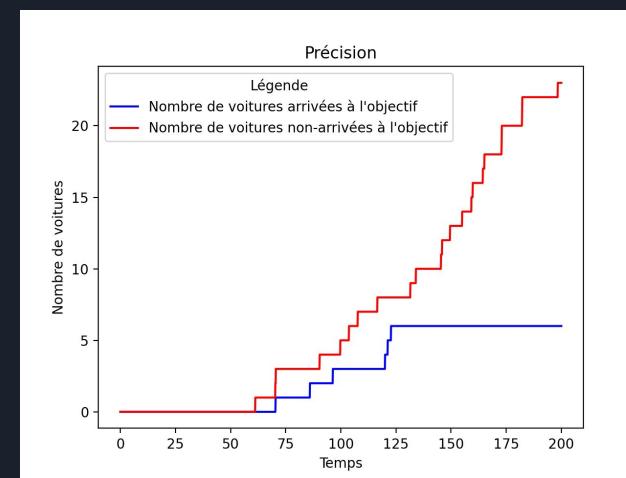
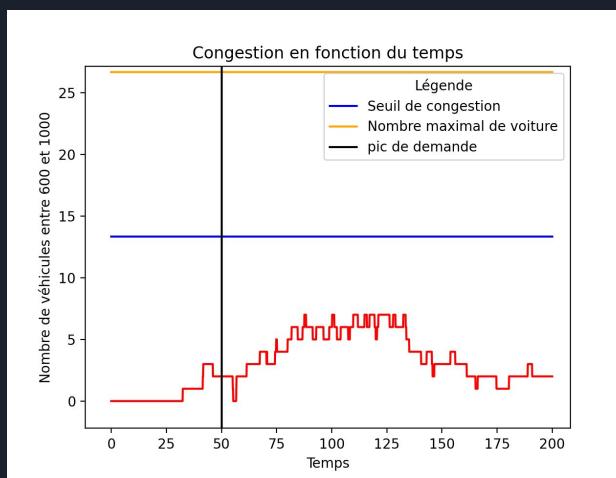
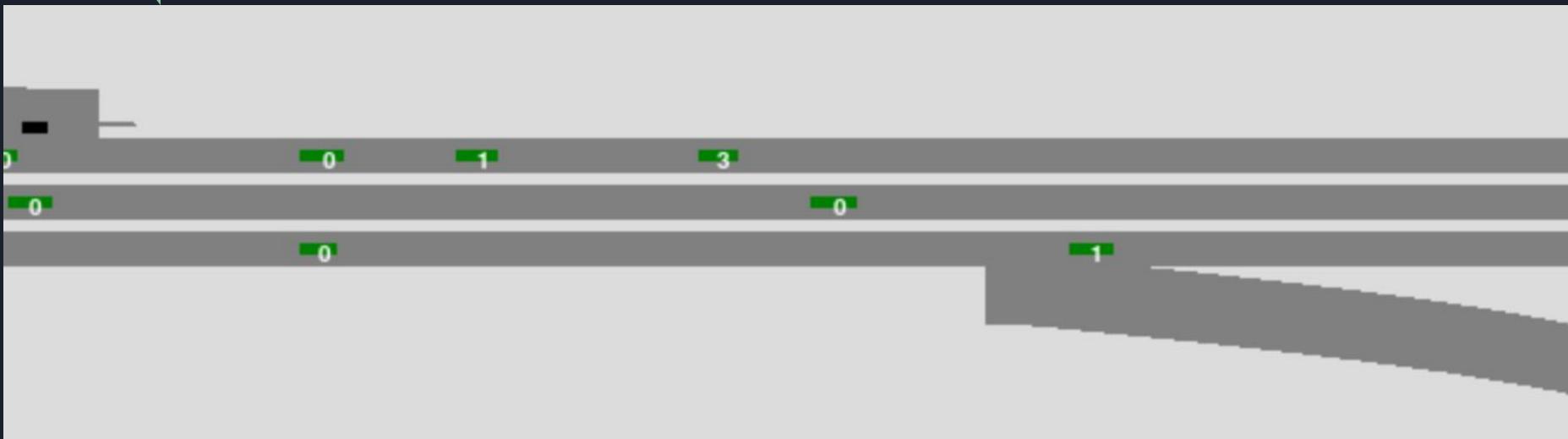
$$\sigma = 30$$

$$\mu = 20$$

$$\sigma = 20$$



Effets de l'altruisme



Attitude contrôlée

Couleur

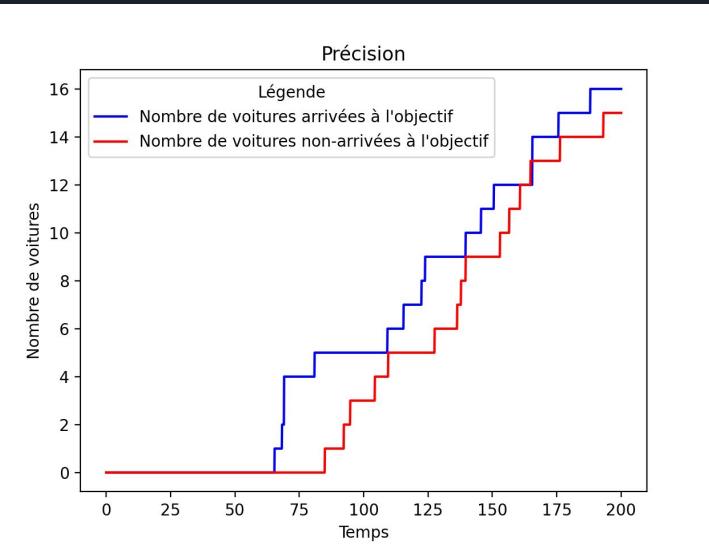
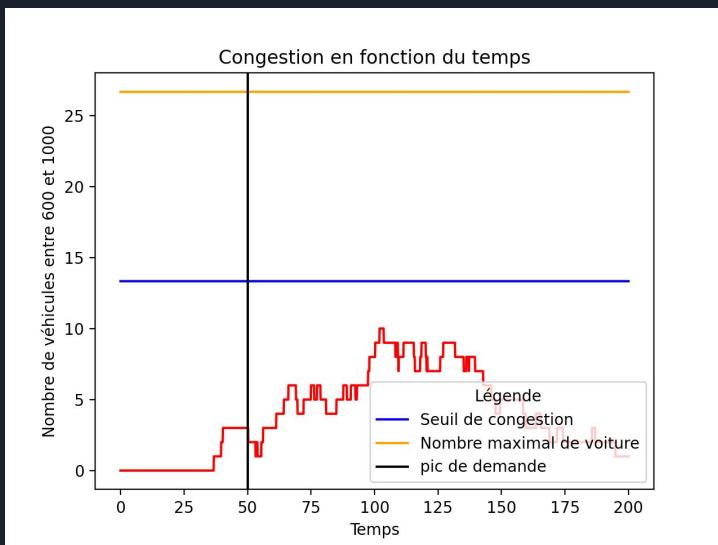
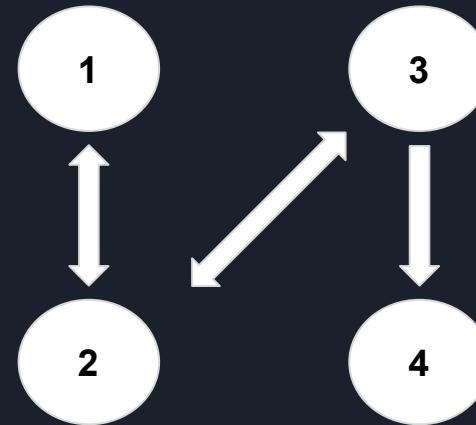
entrée: $v, G = (S, A)$

sortie:modifie le comportement de v

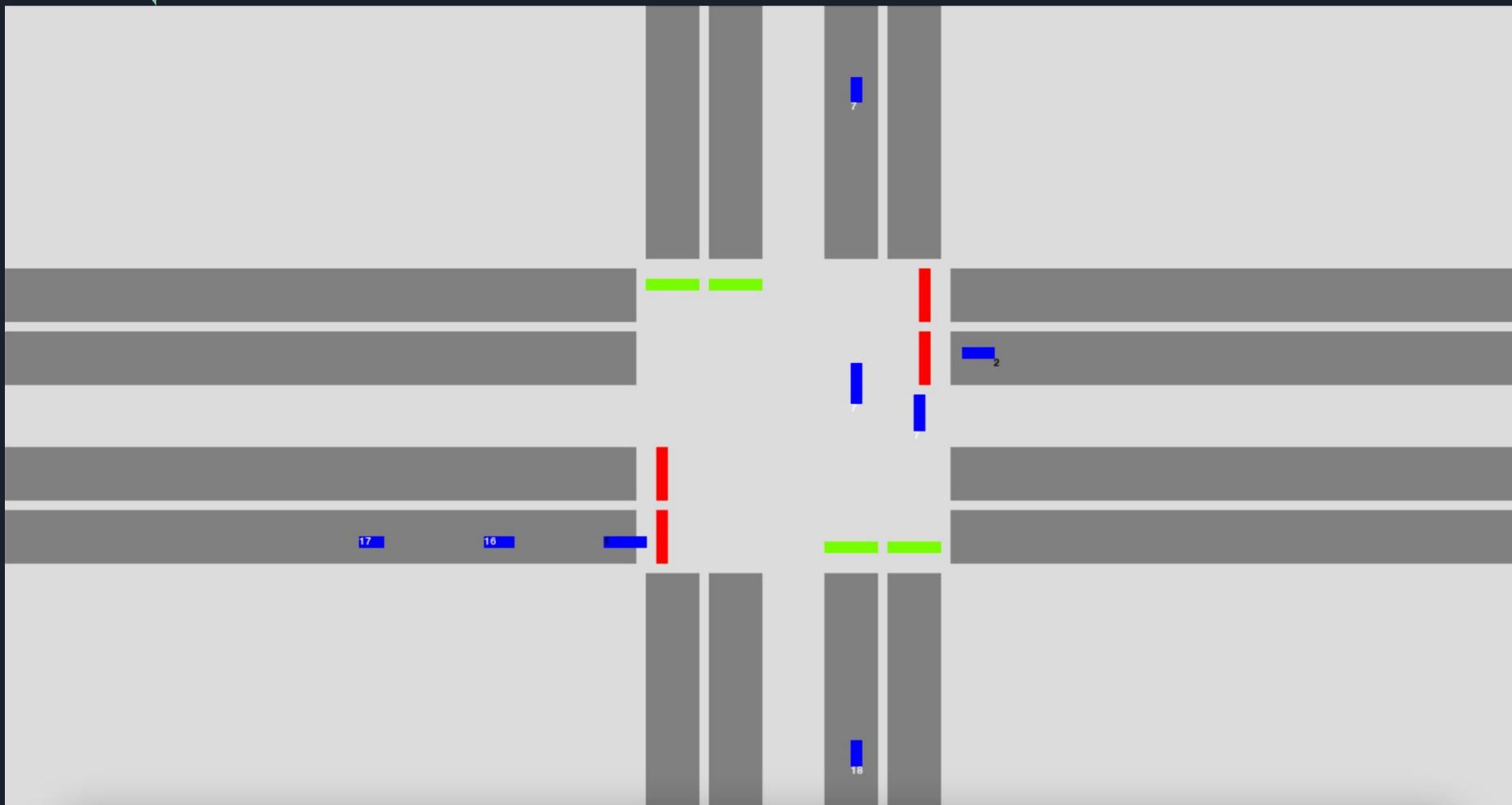
Si $(\text{route}(v), \text{objectif}(v)) \in A :$

v devient égoïste

v devient altruiste



Feux de circulation intelligents



Feux de circulation intelligents

Algorithme 1:(route,t0)

Pour chaque route:

 route.temps=t0

 si route.temps < temps_vert:

 route.feu=vert

 sinon si route.temps < temps_jaune:

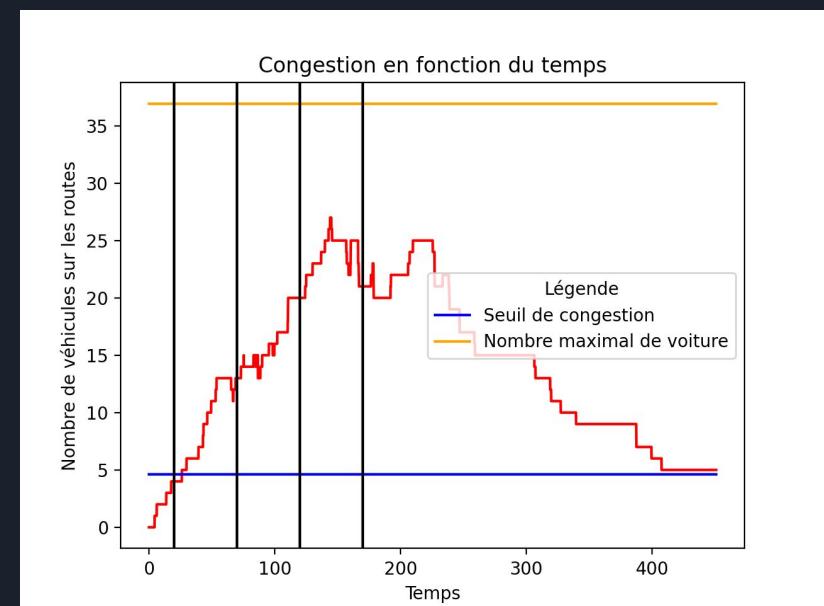
 route.feu=jaune

 sinon si route.temps < temps_rouge:

 route.feu=rouge

 sinon :

 route.temps=0



Feux de circulation intelligents

Algorithme 2:(n,route)

Pour chaque route:

 temps_vert=0

 Pour $i < n$:

 si c_i est activé:

 passer à c_{i+1}

 temps_vert+=10

 Sinon:

 tant que feu opposé est vert:

 attendre

 temps=0

 route.feu=vert

 tant que temps < temps_vert:

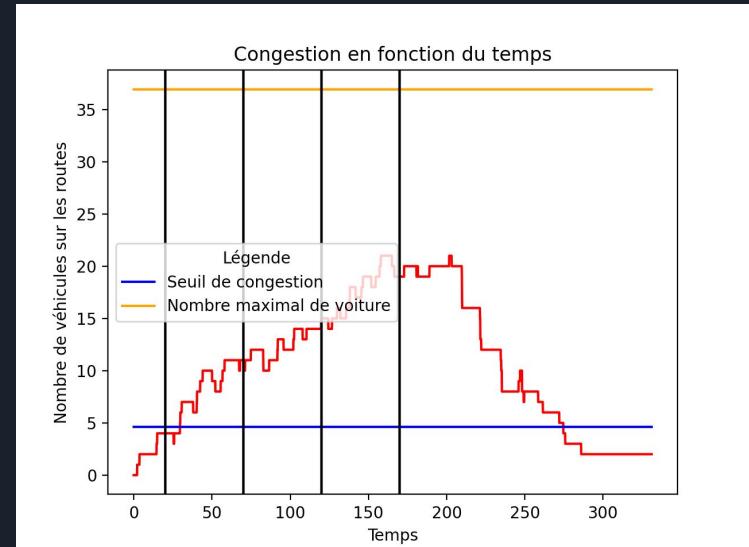
 route.feu=vert

 tant que temps < temps_vert+temps_jaune:

 route.feu=jaune

 tant que temps < temps_vert+temps_jaune+temps_rouge:

 route.feu=rouge



Conclusion

-Long terme: Comportement usager

-Court terme:
Optimisation des feux

```
1 |
2 import pygame, sys
3 import random
4 from pygame.locals import QUIT
5 import numpy as np
6 from math import*
7 import math
8 import numpy.linalg as alg
9 import matplotlib.pyplot as plt
10
11
12
13 #Fonction tournant un rectangle
14 def draw_rotated_rect(surface, rect, angle_radians, color):
15     # Calcul des coins du rectangle
16     top_left = rotate_point(rect.topleft, rect.topleft, angle_radians)
17     top_right = rotate_point(rect.topright, rect.topleft, angle_radians)
18     bottom_right = rotate_point(rect.bottomright, rect.topleft, angle_radians)
19     bottom_left = rotate_point(rect.bottomleft, rect.topleft, angle_radians)
20
21     # Dessin du rectangle orienté
22     vertices = [top_left, top_right, bottom_right, bottom_left]
23     pygame.draw.polygon(surface, color, vertices)
24
25 #Fonction effectuant la rotation d'un point dans le plan, utile pour la fonction précédente
26 def rotate_point(point, origin, angle_radians):
27     # Translation du point relativement à l'origine
28     x, y = point[0] - origin[0], point[1] - origin[1]
29
30     # Rotation du point
31     x_rotated = x * math.cos(angle_radians) - y * math.sin(angle_radians)
32     y_rotated = x * math.sin(angle_radians) + y * math.cos(angle_radians)
33
34     # Translatio du point vers sa position d'origine
35     x_rotated += origin[0]
36     y_rotated += origin[1]
37
38     return round(x_rotated), round(y_rotated)
39
40
41
42
43
44
45
46
47
48
49
```

```
49
50
51 #rgb:255,0,0 couleur des voitures
52
53
54 #pour lancer pygame
55 pygame.init()
56
57
58 #initialiser l'écran
59 SURF = pygame.display.set_mode((1450, 1000))
60 font = pygame.font.SysFont(None, 15)
61
62 color=(220,220,220)
63
64 #paramètres de l'IDM
65 delta=4
66 amax=2
67 vmax=30
68 dmin=100
69 dmin_insertion=70
70 Ti=1
71 b=1.67
72
73 #Paramètres supplémentaires utiles pour la modélisation
74 cste1=80
75 cste2=80
76 alpha=5
77 cste3=50
78 cste4=40
79 d_sortie_min=15
80 cste_sortie=10
81 cste_sortie_bis=1000
82 temps_cooldown=9
83 cste_altruiste=700
84
85
86
87 #Fonctions calculant un tableau des points suivant une courbe de Bézier de points de contrôle pts
88 def tracer_courbe(pts):
89     tab_pts=[]
90     for i in range (1001):
91         tab_pts.append(bezier(*pts, i / 1000))
92     return tab_pts
93
94
95
96
97
98 #Fonction tracant la route et le fond
```

```

97
98 #Fonction traçant la route et le fond
99 def draw():
100     SURF.fill(color)
101
102     pygame.display.set_caption(str(floor(t)))
103
104     for i in range(len(r_set.r_list)):
105         r_set.r_list[i].show_road()
106
107
108
109
110 #Fonction calculant les coordonées d'un point sur une courbe de Bézier de paramètres p0,p1...
111 def bezier(p0,p1,p2,p3,t):
112     px = p0[0]*(1-t)**3 + 3*((1-t)**2)*t*p1[0] + 3*p2[0]*(1-t)*(t**2) + p3[0]*t**3
113     py = p0[1]*(1-t)**2 + 3*((1-t)**2)*t*p1[1] + 3*p2[1]*(1-t)*(t**2) + p3[1]*t**3
114
115     return px, py
116
117
118
119 #Classe permettant de définir les véhicules sur une route de Bézier
120 class vehicle_bezier:
121     def __init__(self,a,v,pos,vmax,coeff,size,p0,p1,p2,p3,time,prev_coeff,arret):
122         #vecteurs de r2
123         self.time=time
124         self.a=a
125         self.v=v
126         self.pos=pos
127         self.vmax=vmax
128         self.coeff=coeff
129         self.size=size
130         self.p0=p0
131         self.p1=p1
132         self.p2=p2
133         self.p3=p3
134         self.prev_coeff=prev_coeff
135         self.arret=arret
136
137
138 #Fonction calculant la position,la vitesse, l'accélération d'un véhicule sur une route de Bézier, en interaction.
139 def update_bezier(self,di,dvi):
140     if (amax*(1-((alg.norm(self.v)/alg.norm(self.vmax))**delta)-((dmin_insertion+alg.norm(self.v)*Ti+((alg.norm(self.v)*dvi)/(2*np.sqrt(amax*b))))/di)**2))>0:
141         self.prev_coeff=self.coeff
142         #Calqué sur l'IDM
143         self.coeff=(amax*(1-((alg.norm(self.v)/alg.norm(self.vmax))**delta)-((dmin+alg.norm(self.v)*Ti+((alg.norm(self.v)*dvi)/(2*np.sqrt(amax*b))))/di)**2))
144         temp=self.pos
145         #Déterminé comme relativement conforme au comportement d'un véhicule sur une voie courbe, peut mal fonctionner selon la courbe.
146         self.pos=bezier(self.p0,self.p1,self.p2,self.p3,((self.coeff*self.time)*100)/100)

```

```
144     temp=self.v
145     #Déterminé comme relativement conforme au comportement d'un véhicule sur une voie courbe, peut mal fonctionner selon la courbe.
146     self.pos=bezier(self.p0,self.p1,self.p2,self.p3,((self.coeff)*self.time)%100)/100
147     temp2=self.v
148     self.v=(np.subtract(temp,self.pos))*dt
149     self.a=(np.subtract(temp2,self.v))*dt
150     self.time+=2.5*dt
151
152     #Fonction calculant la position,la vitesse, l'accélération d'un véhicule sur une route de Bézier, seul.
153     def update_bezier_free(self):
154         self.prev_coeff=self.coeff
155         self.coeff=amax*(1-((alg.norm(self.v))/alg.norm(self.vmax))**delta))
156         temp=self.pos
157         self.pos=bezier(self.p0,self.p1,self.p2,self.p3,((self.coeff)*self.time)%100/100)
158         temp2=self.v
159         self.v=(np.subtract(temp,self.pos))*dt
160         self.a=(np.subtract(temp2,self.v))*dt
161         self.time+=2.5*dt
162
163
164
165
166     #Classe permettant de définir les routes courbes.
167     class road_bezier:
168         def __init__(self,v_list,points):
169             self.v_list=v_list
170             self.points=points
171             self.destination
172
173         #Fonction mettant à jour les paramètres de tous les véhicules sur la route
174         def update_r_bezier(self):
175             if (self.v_list!=[]):
176                 v=self.v_list[0]
177                 if (not(v.arret)):
178                     v.update_bezier_free()
179                 for i in range (len(self.v_list) - 1) :
180                     v1=self.v_list[i+1]
181                     v2=self.v_list[i]
182                     if (not(v1.arret)):
183                         v1.update_bezier((alg.norm(np.subtract(v1.pos,v2.pos))), (alg.norm(np.subtract(v1.v,v2.v))))
184
185         #Fonction affichant les différents véhicules sur la route
186         def show_road_bezier(self):
187             for i in range(0,len(self.v_list)):
188                 v=self.v_list[i]
189                 pygame.draw.rect(SURF,(0,0,255),[v.pos[0] ,v.pos[1],v.size,10])
190
191
192         #Fonction pour sortir de la voie
193         def insertion_bezier(self):
```

```
192     #fonction pour sortir de la voie
193     def insertion_bezier(self):
194         if self.v_list!=[]:
195             v=self.v_list[0]
196             if abs(alg.norm(v.pos)-alg.norm(self.points[3]))<10:
197                 self.destination.insert(0,alg.norm(v.v),alg.norm(v.vmax),alg.norm(v.a),v.size,False,self.destination,False)
198                 self.v_list.remove(v)
199
200
201
202
203     #Fonction arrêtant les véhicules dépassant x (en ordonnée)
204     def stop(self,x):
205         if (self.v_list!=[]):
206             for i in range (len(self.v_list)-1):
207                 v=self.v_list[i]
208                 if (v.pos[1]>=x):
209                     v.arret=True
210
211
212     #Fonction de génération des véhicules sur la voie de bázier
213     def gen_bezier(self):
214         #Vérification de la possibilité de générer un véhicule sans "rentrer dans le dernier" qui se trouve déjà sur la voie
215         if (len(self.v_list)==0 or alg.norm(np.subtract(self.v_list[len(self.v_list)-1].pos,self.points[0]))>30):
216             #Génération d'un nombre aléatoire selon une loi Gaussienne de paramètres (pic,écart_type) donnés plus bas
217             a=np.random.normal(pic,écart_type)
218             if abs(t-a)<5:
219                 v=vehicle_bezier()
220                 v.v=(0,0)
221                 v.a=(0,0)
222                 v.arret=False
223                 vmax_value=30
224                 v.vmax=(vmax_value,vmax_value)
225                 v.rank=len(self.v_list)+1
226                 v.size=random.randint(20,40)/2
227                 v.coeff=1
228                 v.p0=self.points[0]
229                 v.p1=self.points[1]
230                 v.p2=self.points[2]
231                 v.p3=self.points[3]
232                 v.pos=((self.points[0])[0],(self.points[0])[1])
233                 v.cooldown=t
234                 v.time=0
235                 v.prev_coeff=1
236                 i=0
237                 while(i<len(self.v_list)):
238                     self.v_list[i].rank+=1
239                     i+=1
240                 self.v_list.append(v)
241
```

```

242
243     #Insertion d'un véhicule déjà existant de paramètres vitesse,vmax... sur la voie
244     def insert_bezier(self,vitesse,vmax,a,size):
245         v=vehicle_bezier()
246         v.v=(vitesse,0)
247         v.a=(a,0)
248         v.arret=False
249         v.vmax=(vmax,vmax)
250         v.rank=len(self.v_list)+1
251         v.size=size
252         v.coeff=1
253         v.p0=self.points[0]
254         v.p1=self.points[1]
255         v.p2=self.points[2]
256         v.p3=self.points[3]
257         v.pos=((self.points[0])[0],(self.points[0])[1])
258         v.cooldown=t
259         v.time=0
260         v.prev_coeff=1
261         i=0
262         while(i<len(self.v_list)):
263             self.v_list[i].rank+=1
264             i+=1
265         self.v_list.append(v)
266
267
268
269
270     #Classe définissant les différents véhicules sur voies "normales"
271     class vehicle:
272         #paramètres de ta classe
273         def __init__(self,a,v,x,rank,vmax,size,stop,path,cooldown,altruistic):
274             self.cooldown=cooldown
275             self.rank=rank
276             self.x=x
277             self.v=v
278             self.a=a
279             self.vmax=vmax
280             self.size=size
281             self.stop=stop
282             self.path=path #représente une bretelle de sortie ou la route "principale"
283             self.altruistic=altruistic
284
285         #Fonction actualisant x,v,a en interaction.
286         def update_v(self,si,dvi):
287             self.a=amax*(1-(self.v/self.vmax)**delta)-((dmin+self.v*Ti+(self.v*dvi)/(2*np.sqrt(amax*b)))/si)**2
288             if self.v + self.adt < 0:
289                 self.x -= 1/2*self.v*self.v/self.a
290                 self.v = 0
291             else:

```

```
290     self.v = v
291 else:
292     self.v += self.a*dt
293     self.x += self.v*dt + self.a*dt*dt/2
294 if (self.v<=self.vmax/10):
295     self.stop=True
296 else :
297     self.stop=False
298
299
300 #Fonction actualisant x,v,a pour un véhicule libre.
301 def update_free_v(self,route):
302     #Prise en compte de la possibilité d'être dans une zone de feu
303     if route.zone_feu!=[]:
304         zone_arret=route.zone_feu[2]
305         if (self.x<zone_arret[1] and self.x>zone_arret[0]) and (route.zone_feu[0]=="rouge" or route.zone_feu[0]=="jaune") :
306             self.update_v(zone_arret[1]-self.x,self.v)
307         else:
308             self.aamax*(1-(self.v/self.vmax)**delta)
309             self.v+=self.a*dt
310             self.x+=self.v*dt+self.a*dt*dt/2
311             if (self.v<=self.vmax/10):
312                 self.stop=True
313             else :
314                 self.stop=False
315         else:
316             self.aamax*(1-(self.v/self.vmax)**delta)
317             self.v+=self.a*dt
318             self.x+=self.v*dt+self.a*dt*dt/2
319             if (self.v<=self.vmax/10):
320                 self.stop=True
321             else :
322                 self.stop=False
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337 #Classe recensant les différentes routes normales, ou de bézier
338 class road_set:
339     def __init__(self,r_list,r_bezier_list,good_route,bad_route):
```

```
338 class road_set:
339     def __init__(self,r_list,r_bezier_list):
340         self.r_list=r_list
341         self.r_bezier_list=r_bezier_list
342
343
344
345     #Fonction supprimant les véhicules au delà de x=650 (on peut adapter cette valeur)
346     def suppr(self):
347         for i in range(len(self.r_list)):
348             route=self.r_list[i]
349             if route.v_list!=[]:
350                 v=route.v_list[0]
351                 if v.x>650:
352                     route.v_list.remove(v)
353
354
355
356
357     #Fonction affichant les différentes routes
358     def draw_roads(self):
359         for i in range (16):
360             route=self.r_list[i]
361             rectangle=pygame.Rect([route.pt_depart[0],route.pt_depart[1],route.longueur,largeur_routes])
362             draw_rotated_rect(SURF,rectangle,route.angle,(128,128,128))
363         for i in range(len(gen_roads())):
364             route=self.r_list[i]
365             if route.zone_feu!=[]:
366                 if route.zone_feu[0]=="rouge":
367                     couleur_feu=(255,0,0)
368                 elif route.zone_feu[0]=="jaune":
369                     couleur_feu=(255,255,0)
370                 else:
371                     couleur_feu=(124,252,0)
372                 if route.angle==0:
373                     rectangle_feu=pygame.Rect([route.pt_depart[0]+route.longueur+20,route.pt_depart[1],10,largeur_routes])
374                 elif route.angle==np.pi:
375                     rectangle_feu=pygame.Rect([route.pt_depart[0]-route.longueur-20,route.pt_depart[1],10,largeur_routes])
376                 elif route.angle==np.pi/2:
377                     rectangle_feu=pygame.Rect([route.pt_depart[0],route.pt_depart[1]-route.longueur-20,10,largeur_routes])
378                 else:
379                     rectangle_feu=pygame.Rect([route.pt_depart[0],route.pt_depart[1]+route.longueur+20,10,largeur_routes])
380             draw_rotated_rect(SURF,rectangle_feu,route.angle,couleur_feu)
381
382
383
384     #Fonction calculant le comportement des véhicules sur les différentes routes à chaque instant
385     def comportement(self):
386         for i in range (len(self.r_list)):
```

```

583
384     #Fonction calculant le comportement des véhicules sur les différentes routes à chaque instant
385     def comportement(self):
386         for i in range (len(self.r_list)):
387             route=self.r_list[i]
388             for j in range (len(route.v_list)):
389                 v=route.v_list[j]
390                 if (i==0):
391                     if (v.path==self.r_list[1] or (v.path in self.r_bezier_list and v.path.points[0][1]<route.hauteur)):
392                         v.altruistic=False
393                     else :
394                         v.altruistic=True
395                 elif (i==len(self.r_list) - 1):
396                     if (v.path==self.r_list[len(self.r_list)-2] or (v.path in self.r_bezier_list and v.path.points[0][1]>route.hauteur)):
397                         v.altruistic=False
398                     else :
399                         v.altruistic=True
400                 else:
401                     if (v.path==self.r_list[i-1] or v.path==self.r_list[i+1]):
402                         v.altruistic=False
403                     else:
404                         v.altruistic=True
405
406
407     #Fonction mettant à jour les différentes routes
408     def update(self):
409         self.suppr()
410         for i in range(len(self.r_bezier_list)):
411             route1=self.r_bezier_list[i]
412             route1.update_r_bezier()
413             route1.insertion_bezier()
414             for i in range(len(graphe_adjacence)):
415                 self.sortie(self.r_list[i],graphe_adjacence[i][0][0])
416             self.sous_update()
417
418
419     #Bloc de la fonction précédente
420     def sous_update(self):
421         for i in range(len(self.r_list)):
422             route=self.r_list[i]
423             if not(route.v_list==[]):
424                 for j in range (len(route.v_list)):
425                     if (len(route.v_list)>j):
426                         v=route.v_list[j]
427                         #On regarde si il faut doubler pour aller vers son objectif
428                         if (route in gen_roads and v.path in graphe_adjacence[self.r_list.index(route)][1] ):
429                             if i==len(self.r_list)-1 :
430                                 self.doubler_1_route_naif(j,route,self.r_list[i-1],cste1,cste2)
431                             elif (self.r_list[i+1] in gen_roads):
432                                 self.doubler_1_route_naif(i,route,self.r_list[i+1],cste1,cste2)

```

```

417
418
419 #Bloc de la fonction précédente
420 def sous_update(self):
421     for i in range(len(self.r_list)):
422         route=self.r_list[i]
423         if not(route.v_list==[]):
424             for j in range (len(route.v_list)):
425                 if (len(route.v_list)>j):
426                     v=route.v_list[j]
427                     #On regarde si il faut doubler pour aller vers son objectif
428                     if (route in gen_roads and v.path in graphe_adjacence[self.r_list.index(route)][1] ):
429                         if i==len(self.r_list)-1 :
430                             self.doubler_1_route_naif(j,route,self.r_list[i-1],cste1,cste2)
431                         elif (self.r_list[i+1] in gen_roads):
432                             self.doubler_1_route_naif(j,route,self.r_list[i+1],cste1,cste2)
433                         else:
434                             self.doubler_1_route_naif(j,route,self.r_list[i-1],cste1,cste2)
435                     #On regarde si on est au bout de la route
436                     if route in gen_roads and abs(v.x-600)<30:
437                         if v.path in graphe_adjacence[self.r_list.index(route)][0]:
438                             if v.path in self.r_list:
439                                 new_path=self.r_list[(self.r_list.index(v.path)+7) % self.r_list.index(v.path)]
440                                 v.path.insert(0,v.v,v.vmax,v.a,v.size,v.stop,new_path,False)
441                                 route.v_list.remove(v)
442                             else:
443                                 ll=len(graphe_adjacence[i][0])
444                                 ss=random.randint(0,ll-1)
445                                 v.path=graphe_adjacence[i][0][ss]
446                         #On actualise la position,vitesse,accélération
447                     else:
448                         if j==0:
449                             v.update_free_v(route)
450                         else:
451                             v2=route.v_list[j-1]
452                             v.update_v(v2.x-v.x-v.size,abs(v2.v-v.v))
453
454
455
456
457
458
459
460
461
462
463
464
465 #Affichage global (appels au différentes fonctions d'affichage définies avant)
466 def show(self):

```

```

403
404
405 #Affichage global (appels au différentes fonctions d'affichage définies avant)
406 def show(self):
407     SURF.fill(color)
408     pygame.display.set_caption(str(floor(t)))
409     self.draw_roads()
410     for i in range(len(self.r_list)):
411         route=r_set.r_list[i]
412         route.show_road()
413         if route.zone_feu!=[]:
414             #route.update_time()
415             route.update_zone_feu()
416     for i in range (len(r_set.r_bezier_list)):
417         route=r_set.r_bezier_list[i]
418         route.show_road_bezier()
419
420
421
422
423
424 #Génération de véhicules pour toutes les routes où il le faut (appel)
425 def gen(self):
426     for i in range(len(self.r_list)):
427         r=self.r_list[i]
428         if r in gen_roads:
429             r.gen_r(self.r_list.index(r))
430
431
432
433
434 #Fonction calculant la possibilité de doubler ou non et fais doubler dans le cas positif
435 def doubler_1_route_naif(self,j,route1,route2,c1,c2):
436     if (not(j==0)):
437         v2=route1.v_list[j]
438         v1=route1.v_list[j-1]
439         if (abs(v1.x-v2.x)<alpha*(v2.v) and t-v2.cooldown>temps_coldown):
440             k=route2.first_behind(v2.x)
441             if ((route2.free_intervalle((v2.x)-c1,(v2.x)+c2)) ):
442                 route2.insert(v2.x,v2.v,v2.vmax,v2.a,v2.size,v2.stop,v2.path,v2.altruistic)
443                 route1.v_list.remove(v2)
444             else:
445                 v2=route1.v_list[j]
446                 if ((route2.free_intervalle((v2.x)-c1,(v2.x)+c2)) and t-v2.cooldown>temps_coldown ):
447                     route2.insert(v2.x,v2.v,v2.vmax,v2.a,v2.size,v2.stop,v2.path,v2.altruistic)
448                     route1.v_list.remove(v2)
449
450
451
452
453 #Fonction insérant si possible un véhicule de route sur voie_insertion
454 def insertion(self,voie_insertion,route):
455     if (voie_insertion.v_list!=[]):
456         v=voie_insertion.v_list[0]

```



```

530
531
532     #Fonction inséreant un véhicule de voie_de_sortie sur route
533     def sortie(self,route,voie_de_sortie):
534         global good_route
535         global bad_route
536         if (route.v_list!=[]):
537             v_route=route.v_list[route.first_behind(voie_de_sortie.points[0][0]) + 1]
538             if route.angle==0:
539                 y=v_route.x+v_route.size+route.pt_depart[0]
540                 yy=voie_de_sortie.points[0][0]
541             elif route.angle==np.pi:
542                 y=route.pt_depart[0]-v_route.x-v_route.size
543                 yy=voie_de_sortie.points[0][0]
544             elif route.angle==np.pi/2:
545                 y=route.pt_depart[1]+v_route.x+v_route.size
546                 yy=voie_de_sortie.points[0][1]
547             else:
548                 y=route.pt_depart[1]-v_route.x-v_route.size
549                 yy=voie_de_sortie.points[0][1]
550             if (v_route.path==voie_de_sortie):
551                 if len(voie_de_sortie.v_list)>0:
552                     v_sortie=voie_de_sortie.v_list[len(voie_de_sortie.v_list) - 1]
553                     if (alg.norm(np.subtract(v_sortie.pos,voie_de_sortie.points[0]))>d_sortie_min and abs(yy-y)<cste_sortie):
554                         voie_de_sortie.insert_bezier(v_route.v,v_route.vmax,v_route.a,v_route.size)
555                         route.v_list.remove(v_route)
556                         good_route+=1
557                 else:
558                     if (abs(yy-y)<cste_sortie):
559                         voie_de_sortie.insert_bezier(v_route.v,v_route.vmax,v_route.a,v_route.size)
560                         route.v_list.remove(v_route)
561                         good_route+=1
562
563
564
565
566
567
568     #Similaire à doubler
569     def rabattre(self,i,route1,route2,c1,c2):
570         v=route1.v_list[i]
571         if ((route2.free_intervalle((v.x)-c1,(v.x)+c2)) and t-v.cooldown>temps_coldown):
572             route2.insert(v.x,v.v,v.vmax,v.a,v.size,v.stop,v.path,v.altruistic)
573             route1.v_list.remove(v)
574
575
576
577     #or ( (route2.v_list[k].v<v2.v) and (route2.free_intervalle(v2.x,(v2.x)+cste2)))
578
579

```

```
579  
580  
581 #Classe définissant une route  
582 class road:  
583     def __init__(self,v_list,pt_depart,longueur,angle,zone_feu,orientation,pic,temps_vert):  
584         self.v_list=v_list #insérer des véhicules classées par leur rang  
585         self.pt_depart=pt_depart  
586         self.longueur=longueur  
587         self.angle=angle  
588         self.zone_feu=zone_feu  
589         self.orientation=orientation  
590         self.pic=pic  
591         self.temps_vert=temps_vert  
592  
593     #Fonction mettant à jour les différents véhicules de la route  
594     def update_r(self):  
595         if (self.v_list!=[]):  
596             v=self.v_list[0]  
597             v.update_free_v()  
598             for i in range (1,len(self.v_list)) :  
599                 v1=self.v_list[i]  
600                 v2=self.v_list[i-1]  
601                 v1.update_v(v2.x-v1.x-v1.size,abs(v1.v-v2.v))  
602  
603     #Fonction d'affichage de la route et ses véhicules  
604     def show_road(self):  
605         for i in range(len(self.v_list)):  
606             v=self.v_list[i]  
607             if self.angle==0:  
608                 draw_rotated_rect(SURF,pygame.Rect([v.x+v.size+self.pt_depart[0],self.pt_depart[1]+largeur_routes/2,v.size,10]),self.angle,(0,0,255))  
609                 if v.path in r_set.r_list:  
610                     img = font.render(str(r_set.r_list.index(v.path)), True, 'white')  
611                     SURF.blit(img, (v.x+v.size+self.pt_depart[0],self.pt_depart[1]+largeur_routes/2))  
612                 else:  
613                     img = font.render(str(r_set.r_bezier_list.index(v.path)), True, 'black')  
614                     SURF.blit(img, (v.x+v.size+self.pt_depart[0],self.pt_depart[1]+largeur_routes/2))  
615             elif self.angle==np.pi:  
616                 draw_rotated_rect(SURF,pygame.Rect([self.pt_depart[0]-v.x-v.size,self.pt_depart[1]-largeur_routes/2,v.size,10]),self.angle,(0,0,255))  
617                 if v.path in r_set.r_list:  
618                     img = font.render(str(r_set.r_list.index(v.path)), True, 'white')  
619                     SURF.blit(img, (self.pt_depart[0]-v.x-v.size,self.pt_depart[1]-largeur_routes/2))  
620                 else:  
621                     img = font.render(str(r_set.r_bezier_list.index(v.path)), True, 'black')  
622                     SURF.blit(img, (self.pt_depart[0]-v.x-v.size,self.pt_depart[1]-largeur_routes/2))  
623             elif self.angle==np.pi/2:  
624                 draw_rotated_rect(SURF,pygame.Rect([self.pt_depart[0]-largeur_routes/2,self.pt_depart[1]+v.x+v.size,v.size,10]),self.angle,(0,0,255))  
625                 if v.path in r_set.r_list:  
626                     img = font.render(str(r_set.r_list.index(v.path)), True, 'white')  
627                     SURF.blit(img, (self.pt_depart[0]-largeur_routes/2,self.pt_depart[1]+v.x+v.size))  
628             else:
```

```

501     v1.update_v(v2.x-v1.x-v1.size,abs(v1.v-v2.v))
502
503 #Fonction d'affichage de la route et ses véhicules
504 def show_road(self):
505     for i in range(len(self.v_list)):
506         v=self.v_list[i]
507         if self.angle==0:
508             draw_rotated_rect(SURF,pygame.Rect([v.x+v.size+self.pt_depart[0],self.pt_depart[1]+largeur_routes/2,v.size,10]),self.angle,(0,0,255))
509             if v.path in r_set.r_list:
510                 img = font.render(str(r_set.r_list.index(v.path)), True, 'white')
511                 SURF.blit(img, (v.x+v.size+self.pt_depart[0],self.pt_depart[1]+largeur_routes/2))
512             else:
513                 img = font.render(str(r_set.r_bezier_list.index(v.path)), True, 'black')
514                 SURF.blit(img, (v.x+v.size+self.pt_depart[0],self.pt_depart[1]+largeur_routes/2))
515         elif self.angle==np.pi:
516             draw_rotated_rect(SURF,pygame.Rect([self.pt_depart[0]-v.x-v.size,self.pt_depart[1]-largeur_routes/2,v.size,10]),self.angle,(0,0,255))
517             if v.path in r_set.r_list:
518                 img = font.render(str(r_set.r_list.index(v.path)), True, 'white')
519                 SURF.blit(img, (self.pt_depart[0]-v.x-v.size,self.pt_depart[1]-largeur_routes/2))
520             else:
521                 img = font.render(str(r_set.r_bezier_list.index(v.path)), True, 'black')
522                 SURF.blit(img, (self.pt_depart[0]-v.x-v.size,self.pt_depart[1]-largeur_routes/2))
523         elif self.angle==np.pi/2:
524             draw_rotated_rect(SURF,pygame.Rect([self.pt_depart[0]-largeur_routes/2,self.pt_depart[1]+v.x+v.size,v.size,10]),self.angle,(0,0,255))
525             if v.path in r_set.r_list:
526                 img = font.render(str(r_set.r_list.index(v.path)), True, 'white')
527                 SURF.blit(img, (self.pt_depart[0]-largeur_routes/2,self.pt_depart[1]+v.x+v.size))
528             else:
529                 img = font.render(str(r_set.r_bezier_list.index(v.path)), True, 'black')
530                 SURF.blit(img, (self.pt_depart[0]-largeur_routes/2,self.pt_depart[1]+v.x+v.size))
531         else:
532             draw_rotated_rect(SURF,pygame.Rect([self.pt_depart[0]+largeur_routes/2,self.pt_depart[1]-v.x-v.size,v.size,10]),self.angle,(0,0,255))
533             if v.path in r_set.r_list:
534                 img = font.render(str(r_set.r_list.index(v.path)), True, 'white')
535                 SURF.blit(img, (self.pt_depart[0]+largeur_routes/2,self.pt_depart[1]-v.x-v.size))
536             else:
537                 img = font.render(str(r_set.r_bezier_list.index(v.path)), True, 'black')
538                 SURF.blit(img, (self.pt_depart[0]+largeur_routes/2,self.pt_depart[1]-v.x-v.size))
539
540
541
542
543
544
545
546 #FOnction supprimant les véhicules au dela de x
547 def suppr_r(self,x):
548     global good_route
549     global bad_route
550     for i in range(len(self.v_list)-1):

```

```
643
644
645
646 #FOnction supprimant les véhicules au dela de x
647 def suppr_r(self,x):
648     global good_route
649     global bad_route
650     for i in range (len(self.v_list)-1):
651         v=self.v_list[i]
652         if (v.x>x):
653             self.v_list.remove(v)
654             if (v.path==self):
655                 good_route+=1
656             else:
657                 bad_route+=1
658
659 #Fonction regardant si l'intervalle [a;b] est libre de véhicules
660 def free_intervalle(self,a,b):
661     #a<b
662     if (self.v_list==[]):
663         return True
664     else:
665         i=len(self.v_list)-1
666         v=self.v_list[i]
667         while(v.x<a and i>=0):
668             i=i-1
669             v=self.v_list[i]
670         if i== -1:
671             return True
672         elif v.x<b:
673             return False
674     return True
675
676 #Fonction calculant le nombre de véhicules sur [a;b]
677 def num_voiture(self,a,b):
678     if self.v_list==[]:
679         return 0
680     else:
681         count=0
682         i=0
683         v=self.v_list[0]
684         while(i<len(self.v_list)):
685             v=self.v_list[i]
686             if (v.x>a and v.x < b):
687                 count=count+1
688             i=i+1
689     return count
690
691 #Fonction insérant un véhicule à un endroit aléatoire sur la route (utile pour les tests uniquement)
692 def insert_rand(self):
```

```
090
691     #Fonction insérant un véhicule à un endroit aléatoire sur la route (utile pour les tests uniquement)
692 def insert_rand(self):
693     v=vehicle()
694     v.v=20/2
695     v.a=5/2
696     v.vmax=random.randint(20,30)/2
697     i=random.randint(0,len(self.v_list)-2)
698     v.rank=i
699     v.size=random.randint(20,40)/2
700     if (self.v_list[i].x-self.v_list[i+1].x-self.v_list[i].size>30/2):
701         v.x=random.randint(floor(self.v_list[i+1].x-self.v_list[i+1].size+10/2),ceil(self.v_list[i].x+v.size-10/2))
702         j=len(self.v_list)-1
703         while (j>0 and self.v_list[j].x<v.x):
704             j=j-1
705         self.v_list.insert(j+1,v)
706
707     #Fonction pour insérer à la position x un véhicule de paramètres vitesse,vmax,a,size...
708 def insert(self,x,vitesse,vmax,a,size,stop,path,altruistic):
709     v=vehicle()
710     v.v=vitesse
711     v.a=a
712     v.vmax=vmax
713     v.x=x
714     v.size=size
715     vcooldown=t
716     v.stop=stop
717     v.path=path
718     v.altruistic=altruistic
719     j=len(self.v_list)-1
720     while (j>=0 and self.v_list[j].x<v.x):
721         j=j-1
722     v.rank=j
723     if (j== -1):
724         self.v_list.insert(0,v)
725     else:
726         self.v_list.insert(j+1,v)
727
728     #Fonction calculant le premier véhicule derrière x
729 def first_behind(self,x):
730     if (self.v_list==[]):
731         return 0
732     else:
733         i=len(self.v_list)-1
734         v=self.v_list[i]
735         while(i>0 and v.x<x):
736             i=i-1
737             v=self.v_list[i]
738         return i-1
739
```

```
740
741     #Fonction calculant le premier véhicule devant x+c
742     def first_infront(self,x,c):
743         #c est utile pour l'altruisme
744         if (self.v_list==[]):
745             print("erreur de firstinfront")
746             return 0
747         else:
748             i=0
749             v=self.v_list[i]
750             while(i+1<len(self.v_list) and v.x-x>c):
751                 i=i+1
752                 v=self.v_list[i]
753             return i-1
754
755     #Fonction regardant si un véhicule est arrêté dans l'intervalle [a;b]
756     def is_stopped_in_intervalle(self,a,b):
757         if self.v_list==[]:
758             return False
759         else:
760             i=0
761             v=self.v_list[0]
762             while(i<len(self.v_list)):
763                 v=self.v_list[i]
764                 if (v.x>a and v.x < b and v.stop):
765                     return True
766                 i=i+1
767             return False
768
769
770
771
772     #Fonction de génération des véhicules sur la route
773     def gen_r(self,rang_route):
774         if self.free_intervalle(-40,30):
775             a=np.random.normal(self.pic,ecart_type)
776             if abs(t-a)<5:
777                 v=vehicle()
778                 v.v=20/2
779                 v.a=5/2
780                 v.vmax=30
781                 v.rank=len(self.v_list)+1
782                 v.size=random.randint(20,40)
783                 v.x=-v.size
784                 vcooldown=t
785                 v.stop=False
786                 l=len(graphe_adjacence[rang_route])
787                 s=random.randint(0,l-1)
788                 ll=len(graphe_adjacence[rang_route][s])
```

```

//1
772 #Fonction de génération des véhicules sur la route
773 def gen_r(self,rang_route):
774     if self.free_intervalle(-40,30):
775         a=np.random.normal(self.pic,ecart_type)
776         if abs(t-a)<5:
777             v=vehicle()
778             v.v=20/2
779             v.a=5/2
780             v.vmax=30
781             v.rank=len(self.v_list)+1
782             v.size=random.randint(20,40)
783             v.x=-v.size
784             v.cooldown=t
785             v.stop=False
786             l=len(graphe_adjacence[rang_route])
787             s=random.randint(0,l-1)
788             ll=len(graphe_adjacence[rang_route][s])
789             ss=random.randint(0,ll-1)
790             v.path=graphe_adjacence[rang_route][s][ss]
791             r=random.randint(1,2)
792             if (r==1):
793                 v.altruistic=True
794             else:
795                 v.altruistic=False
796             i=0
797             while(i<len(self.v_list)):
798                 self.v_list[i].rank+=1
799                 i+=1
800             self.v_list.append(v)

801
802
803 #Fonction mettant à jour le feu de la route s'il y en a un
804 def update_zone_feu(self):
805     time=self.zone_feu[1]
806     if time<self.temps_vert:
807         self.zone_feu[0]="vert"
808         if not(pause):
809             self.zone_feu[1]+=dt
810     elif (time<self.temps_vert+temps_jaune):
811         self.zone_feu[0]="jaune"
812         if not(pause):
813             self.zone_feu[1]+=dt
814     elif (time<self.temps_vert+temps_jaune+temps_rouge):
815         self.zone_feu[0]="rouge"
816         if not(pause):
817             self.zone_feu[1]+=dt
818     else:
819         self.zone_feu[1]=0
820

```

```
820
821
822 #Fonction utile pour la précédente
823 def update_time(self):
824     if (r_set.r_list.index(self) % 2 == 0):
825         if (not(r_set.r_list[(r_set.r_list.index(self)+2) % len(gen_roads)].zone_feu[0]=="vert" or \
826                 r_set.r_list[(r_set.r_list.index(self)+3) % len(gen_roads)].zone_feu[0]=="vert" or \
827                 r_set.r_list[(r_set.r_list.index(self)-1) % len(gen_roads)].zone_feu[0]=="vert" or \
828                 r_set.r_list[(r_set.r_list.index(self)-2) % len(gen_roads)].zone_feu[0]=="vert")):
829             self.temps_vert=self.num_voiture(0,600)*10
830             self.zone_feu[1]=0
831         else:
832             self.temps_vert=0
833             self.zone_feu[1]=temps_jaune
834     else:
835         if (not(r_set.r_list[(r_set.r_list.index(self)+1) % len(gen_roads)].zone_feu[0]=="vert" or \
836                 r_set.r_list[(r_set.r_list.index(self)+2) % len(gen_roads)].zone_feu[0]=="vert" or \
837                 r_set.r_list[(r_set.r_list.index(self)-2) % len(gen_roads)].zone_feu[0]=="vert" or \
838                 r_set.r_list[(r_set.r_list.index(self)-3) % len(gen_roads)].zone_feu[0]=="vert")):
839             self.temps_vert=self.num_voiture(0,600)*10
840             self.zone_feu[1]=0
841         else:
842             self.temps_vert=0
843             self.zone_feu[1]=temps_jaune
844
845
846
847
848
849
850
851
852
853
854
855 t=0
856 et=0
857 dt=0.1
858
859
860 entree=0
861 entree=int(entree)
862 dx=600
863 flow=[]
864 cste_congestion=[]
865 cste_capamax=[]
866 esp_taille_voiture=30
867
868 good_route=0
869 bad_route=0
```

```
00/
868 good_route=0
869 bad_route=0
870 good_route_tab=[]
871 bad_route_tab=[]
872
873 #####
874
875
876
877 temps_rouge=40
878 temps_jaune=20
879
880
881
882 pic=50
883 ecart_type=20
884 largeur_routes=50
885
886 pts_entree=[]
887 pts_sortie=[]
888
889 r_set=road_set()
890 r1=road()
891 r2=road()
892 r3=road()
893 r4=road()
894 r5=road()
895 r6=road()
896 r7=road()
897 r8=road()
898 r9=road()
899 r10=road()
900 r11=road()
901 r12=road()
902 r13=road()
903 r14=road()
904 r15=road()
905 r16=road()
906
907 r1.pt_depart=(600,300)
908 r2.pt_depart=(600,460-100)
909 r3.pt_depart=(0,520-100)
910 r4.pt_depart=(0,580-100)
911 r5.pt_depart=(660,640-100)
912 r6.pt_depart=(720,640-100)
913 r7.pt_depart=(780,1240-100)
914 r8.pt_depart=(840,1240-100)
915 r9.pt_depart=(900,580-100)
916 r10.pt_depart=(900,520-100)
```

```
900
907 r1.pt_depart=(600,300)
908 r2.pt_depart=(600,460-100)
909 r3.pt_depart=(0,520-100)
910 r4.pt_depart=(0,580-100)
911 r5.pt_depart=(660,640-100)
912 r6.pt_depart=(720,640-100)
913 r7.pt_depart=(780,1240-100)
914 r8.pt_depart=(840,1240-100)
915 r9.pt_depart=(900,580-100)
916 r10.pt_depart=(900,520-100)
917 r11.pt_depart=(1500,460-100)
918 r12.pt_depart=(1500,400-100)
919 r13.pt_depart=(840,340-100)
920 r14.pt_depart=(780,340-100)
921 r15.pt_depart=(720,340-600-100)
922 r16.pt_depart=(660,340-600-100)
923
924 r1.v_list=[]
925 r2.v_list=[]
926 r3.v_list=[]
927 r4.v_list=[]
928 r5.v_list=[]
929 r6.v_list=[]
930 r7.v_list=[]
931 r8.v_list=[]
932 r9.v_list=[]
933 r10.v_list=[]
934 r11.v_list=[]
935 r12.v_list=[]
936 r13.v_list=[]
937 r14.v_list=[]
938 r15.v_list=[]
939 r16.v_list=[]
940
941
942
943
944
945 r_b1_1=road_bezier()
946 r_b1_2=road()
947 r_b2_1=road_bezier()
948 r_b2_2=road()
949 r_b3_1=road_bezier()
950 r_b3_2=road()
951 r_b4_1=road_bezier()
952 r_b4_2=road()
953 r_b5_1=road_bezier()
954 r_b5_2=road()
955 r_b6_1=road_bezier()
```

```
944
945 r_b1_1=road_bezier()
946 r_b1_2=road()
947 r_b2_1=road_bezier()
948 r_b2_2=road()
949 r_b3_1=road_bezier()
950 r_b3_2=road()
951 r_b4_1=road_bezier()
952 r_b4_2=road()
953 r_b5_1=road_bezier()
954 r_b5_2=road()
955 r_b6_1=road_bezier()
956 r_b6_2=road()
957 r_b7_1=road_bezier()
958 r_b7_2=road()
959 r_b8_1=road_bezier()
960 r_b8_2=road()
961
962
963
964
965 r_b1_2.angle=0
966 r_b2_2.angle=0
967 r_b3_2.angle=-np.pi/2
968 r_b4_2.angle=-np.pi/2
969 r_b5_2.angle=np.pi
970 r_b6_2.angle=np.pi
971 r_b7_2.angle=np.pi/2
972 r_b8_2.angle=np.pi/2
973
974
975 #longueur=300
976 r_b1_2.pt_depart=(600,420)
977 r_b2_2.pt_depart=(600,480)
978 r_b3_2.pt_depart=(780,540)
979 r_b4_2.pt_depart=(840,540)
980 r_b5_2.pt_depart=(900,360)
981 r_b6_2.pt_depart=(900,300)
982 r_b7_2.pt_depart=(720,-360+600)
983 r_b8_2.pt_depart=(660,-360+600)
984
985 r_b1_2.longueur=300
986 r_b2_2.longueur=300
987 r_b3_2.longueur=300
988 r_b4_2.longueur=300
989 r_b5_2.longueur=300
990 r_b6_2.longueur=300
991 r_b7_2.longueur=300
992 r_b8_2.longueur=300
993
```

```
993
994 r_b1_2.v_list=[]
995 r_b2_2.v_list=[]
996 r_b3_2.v_list=[]
997 r_b4_2.v_list=[]
998 r_b5_2.v_list=[]
999 r_b6_2.v_list=[]
1000 r_b7_2.v_list=[]
1001 r_b8_2.v_list=[]
1002
1003
1004
1005 r_b1_1.points=[(600,440),(600+130,440-150),(-600+200,440-150),(600+200,440-200)]
1006 r_b1_1.v_list=[]
1007 r_b1_1.destination=r13
1008 r_b2_1.points=[(600,500),(650,400),(650,600),(-635,540)]
1009 r_b2_1.v_list=[]
1010 r_b2_1.destination=r13
1011
1012 r_b3_1.points=[(800,540),(800-130,540-350),(-800-230,540-250),(600,340)]
1013 r_b3_1.v_list=[]
1014 r_b4_1.points=[(860,540),(850,400),(880,480),(-900,500)]
1015 r_b4_1.v_list=[]
1016
1017 r_b5_1.points=[(900,340),(800,300),(700,400),(700,540)]
1018 r_b5_1.v_list=[]
1019 r_b6_1.points=[(900,275),(880,190),(870,220),(865,240)]
1020 r_b6_1.v_list=[]
1021
1022 r_b7_1.points=[(700,240),(775,320),(850,420),(900,440)]
1023 r_b7_1.v_list=[]
1024 r_b8_1.points=[(630,240),(620,250),(610,260),(600,275)]
1025 r_b8_1.v_list=[]
1026
1027 r_set.r_bezier_list=[r_b1_1,r_b2_1,r_b3_1,r_b4_1,r_b5_1,r_b6_1,r_b7_1,r_b8_1]
1028
1029 r_b1_1.destination=r14
1030 r_b2_1.destination=r5
1031 r_b3_1.destination=r2
1032 r_b4_1.destination=r9
1033 r_b5_1.destination=r6
1034 r_b6_1.destination=r13
1035 r_b7_1.destination=r10
1036 r_b8_1.destination=r1
1037
1038
1039
1040
1041 r_set.r_list=[r3,r4,r7,r8,r11,r12,r15,r16,r1,r2,r5,r6,r9,r10,r13,r14,r_b1_2,r_b2_2,r_b3_2,r_b4_2,r_b5_2,r_b6_2,r_b7_2,r_b8_2 ]
1042
```

```
1042
1043
1044 angles_list=[0,0,-np.pi/2,-np.pi/2,np.pi,np.pi,np.pi/2,np.pi/2,np.pi,np.pi,np.pi/2,np.pi/2,0,0,-np.pi/2,-np.pi/2]
1045 for i in range (16):
1046     r_set.r_list[i].longueur=600
1047     r_set.r_list[i].angle=angles_list[i]
1048
1049
1050
1051
1052 #0->r3,1->r4,2->r7,3->r8,4->r11,5->r12,6->r15,7->r16
1053 graphe_adjacence=[
1054     [[r_b1_1,r_b1_2],[r_b2_1,r_b2_2]],[[r_b2_1,r_b2_2],[r_b1_1,r_b1_2]],\
1055     [[r_b3_1,r_b3_2],[r_b4_1,r_b4_2]],[[r_b4_1,r_b4_2],[r_b3_1,r_b3_2]],\
1056     [[r_b5_1,r_b5_2],[r_b6_1,r_b6_2]],[[r_b6_1,r_b6_2],[r_b5_1,r_b5_2]],\
1057     [[r_b7_1,r_b7_2],[r_b8_1,r_b8_2]],[[r_b8_1,r_b8_2],[r_b7_1,r_b7_2]]\
1058 ]
1059 ]
1060
1061 gen_roads=[r3,r4,r7,r8,r11,r12,r15,r16]
1062
1063
1064 diff=20
1065 for i in range(len(gen_roads)):
1066     r_set.r_list[i].pic=diff
1067     if i%2==1:
1068         diff+=50
1069
1070
1071 for i in range (len(gen_roads)):
1072     r_set.r_list[i].temps_vert=20
1073     r_set.r_list[0].zone_feu=["rouge",0,(400,600)]
1074     r_set.r_list[1].zone_feu=["rouge",0,(400,600)]
1075     r_set.r_list[4].zone_feu=["rouge",0,(400,600)]
1076     r_set.r_list[5].zone_feu=["rouge",0,(400,600)]
1077     r_set.r_list[2].zone_feu=["rouge",tempo_jaune,(400,600)]
1078     r_set.r_list[3].zone_feu=["rouge",tempo_jaune,(400,600)]
1079     r_set.r_list[6].zone_feu=["rouge",tempo_jaune,(400,600)]
1080     r_set.r_list[7].zone_feu=["rouge",tempo_jaune,(400,600)]
1081
1082
1083 for i in range(len(gen_roads),len(r_set.r_list)):
1084     r_set.r_list[i].zone_feu=[]
1085
1086
1087 #####
1088 #####
1089 #####
1090 #####
1091 pause=True
```

```
1090
1091 pause=True
1092 run=True
1093 while (run):
1094     pygame.time.delay(20)
1095     for event in pygame.event.get():
1096         if event.type==pygame.KEYDOWN:
1097             if event.key==pygame.K_ESCAPE:
1098                 run=False
1099             if event.type == QUIT:
1100                 pygame.quit()
1101                 sys.exit()
1102             if event.type==pygame.KEYDOWN:
1103                 if event.key==pygame.K_j:
1104                     pause=not(pause)
1105
1106
1107 #x,z=pygame.mouse.get_pos()
1108 #print(x,z)
1109 for event in pygame.event.get():
1110     if event.type == QUIT:
1111         pygame.quit()
1112         sys.exit()
1113
1114
1115
1116 if not(pause):
1117     if t-et>=0.5:
1118         r_set.gen()
1119         et+=0.5
1120     r_set.update()
1121     t=t+dt
1122     nb_voitures=0
1123     for i in range(len(gen_roads)):
1124         nb_voitures+=r_set.r_list[i].num_voiture(entree,entree+dx)
1125     flow.append(nb_voitures)
1126     cste_congestion.append(dx/(esp_taille_voiture+dmin))
1127     cste_capamax.append(len(gen_roads)*dx/(esp_taille_voiture+dmin))
1128 r_set.show()
1129 pygame.display.update()
1130 print(r3.temps_vert,r3.zone_feu[1])
1131
1132
1133
1134
1135
1136
1137 #Affichage du graphe de congestion
1138 time=np.linspace(0,floor(t),floor(t/dt))
1139 if (len(time)>len(flow)):
```

```
1135  
1136 #Affichage du graphe de congestion  
1137 time=np.linspace(0,floor(t),floor(t/dt))  
1138 if (len(time)>len(flow)):  
1139     time=time[:-1]  
1140 elif (len(flow)>len(time)):  
1141     del flow [-1]  
1142     del cste_congestion [-1]  
1143     del cste_capamax [-1]  
1144 plt.plot(time,flow,color='red')  
1145 plt.plot(time,cste_congestion,color='blue',label="Seuil de congestion")  
1146 plt.plot(time,cste_capamax,color='orange',label="Nombre maximal de voiture")  
1147 plt.ylabel("Nombre de véhicules sur les routes")  
1148 plt.xlabel("Temps")  
1149 for i in range (len(gen_roads)):  
1150     plt.axvline(x=r_set.r_list[i].pic,color="black")  
1151 plt.legend(title="Légende")  
1152 plt.title("Congestion en fonction du temps")  
1153 plt.show()  
1154  
1155  
1156
```