

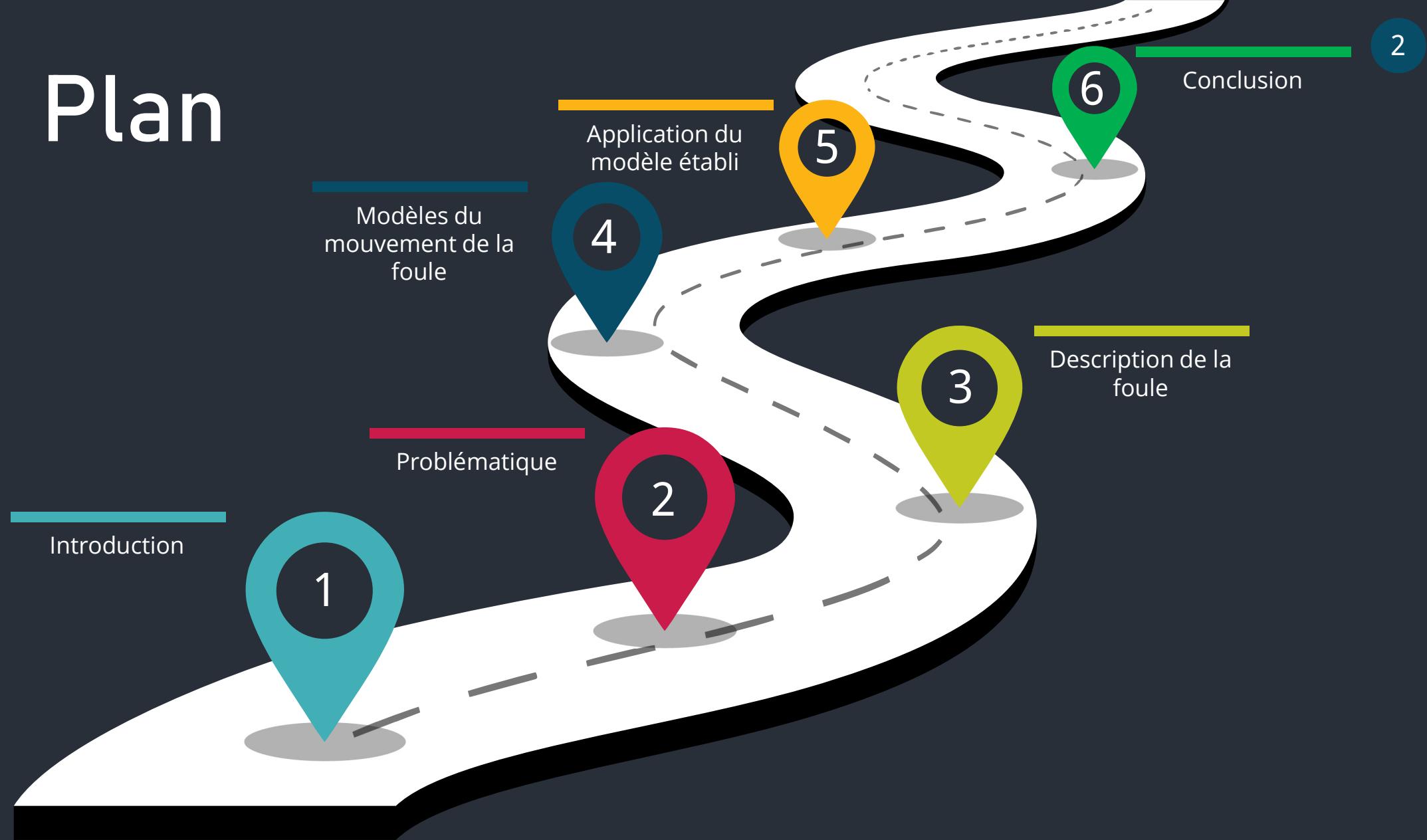


MODELISATION DU MOUVEMENT DES FOULES DE PIETONS

Réalisé par : EL BARTHICHI MOHAMMED

Encadré par : LACHHAB ABDESLAM

Plan





1- introduction



Figure 1 : Une bousculade qui a causé la mort de 2400 personnes



Figure 2 : Situation réelle d'évacuation d'un terrain de football



Figure 3 : Explosion d'une tour qui a causé l'évacuation 17 000 personnes de 90 étages

2- Problématique

Les lois qui gèrent le mouvement des individus doivent être capables de bien décrire le comportement des foules en toutes situations. Alors :

Comment modéliser une foule de piétons en situations de panique de façon à obtenir des résultats semblables à ceux réels ?





3- Description de la foule

Une foule est une multitude de piétons regroupées dans un même lieu. Chaque piéton :

- Admet une vitesse souhaitée.
- Admet une direction souhaitée.
- Est adepte au principe du moindre effort.

✓ En situation de panique, les piétons :

- Devient nerveux, ce qui induit un comportement autiste.
- Se déplacent plus vite que la normale.
- Interagissent plus fréquemment avec les autres piétons et les obstacles.

✓ Si la vitesse souhaitée est supérieure à 1,5m/s, il y a blocage des individus

Les valeurs moyennes de la vitesse souhaitée:

Auteur	Vitesse souhaitée moyenne
L.F. Henderson (1971)	1,34 m/s
Dirk Helbing (2000)	1,3 m/s
Mehdi Moussaïd (2010)	1,1 m/s – 1,48 m/s

4- Modèles de la foule



Modélisation
de la foule

Modèles
macroscopiques

Modèles
microscopiques

Modèle des forces sociales

- ✓ Les interactions des individus au sein d'une foule sont :
 - Force motrice : force distance
 - Interaction piéton-piéton : force de contact
 - Interaction piéton-obstacle : force de contact
- ✓ Chaque piéton est en interaction avec tous les piétons et les obstacles qui le sont voisins

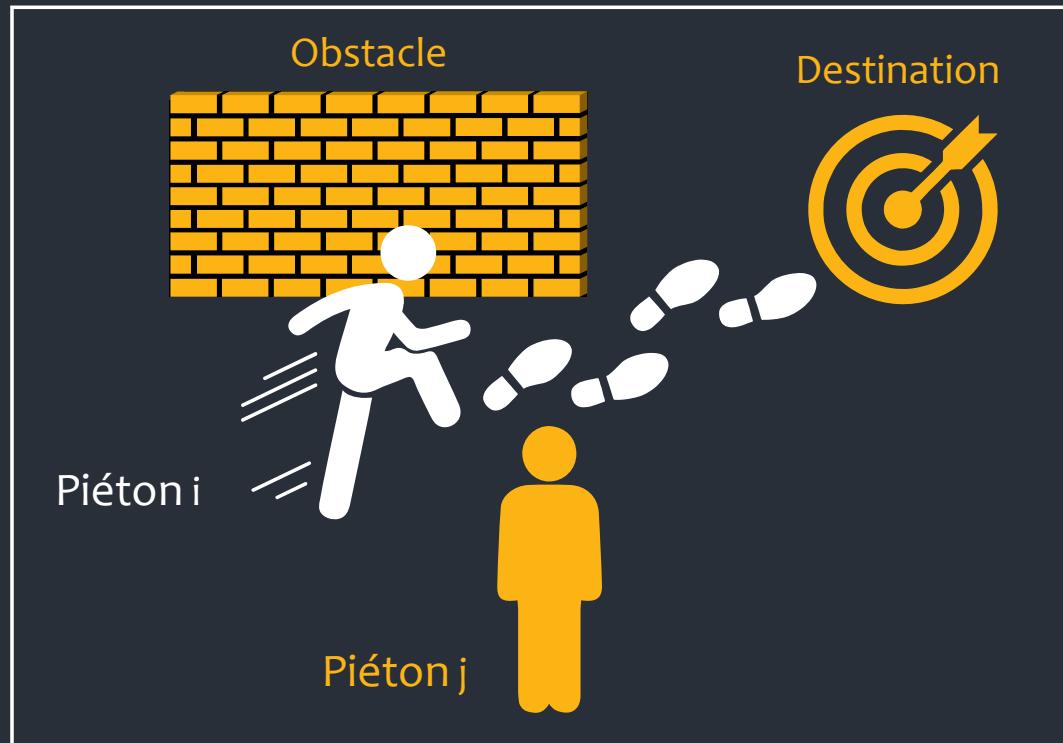


Figure 4 : Schéma représentant les différentes interactions d'un piéton au sein d'une foule

Modèle des forces sociales

- ✓ Le modèle 2D discret permet de simplifier la représentation des forces qui s'exercent sur chaque individu.
- ✓ Chaque piéton ‘i’ est représenté par un disque de rayon r_i , de une masse m_i , et d'une vitesse souhaitée v_i^0

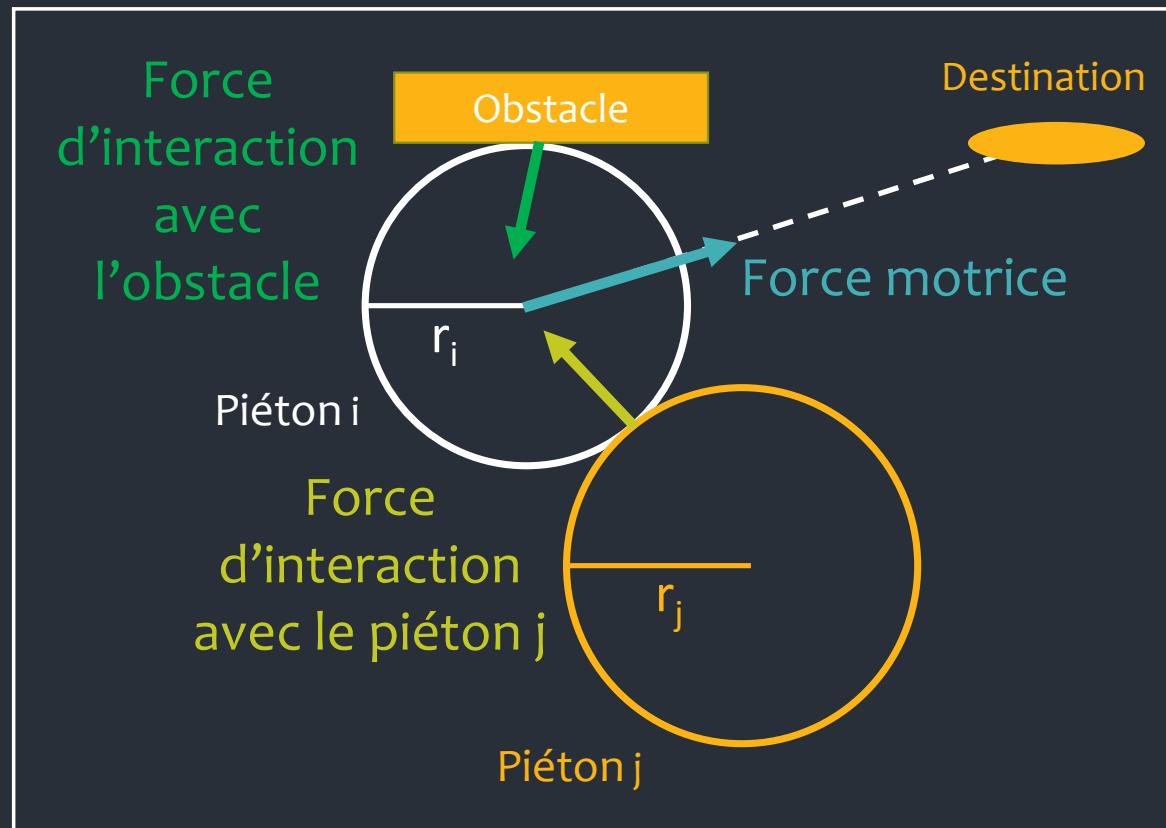


Figure 5 : Schéma simplifié représentant les différentes forces sociales

Modèle des forces sociales

□ Force motrice :

$$\vec{f}_i = m_i \frac{\nu_i^0 \cdot \vec{e}_i - \overrightarrow{\nu_i(t)}}{\tau_i}$$

Avec:

m_i : la masse du piéton i.

ν_i^0 : la vitesse désirée du piéton i.

\vec{e}_i : la direction souhaitée du piéton i

$\overrightarrow{\nu_i(t)}$: la vitesse réelle du piéton i

τ_i : temps de relaxation (temps dans lequel le piéton retrouve sa vitesse désirée après contact)

On prend $\tau_i = 0,5\text{s}$ (d'après Dirk Helbing)

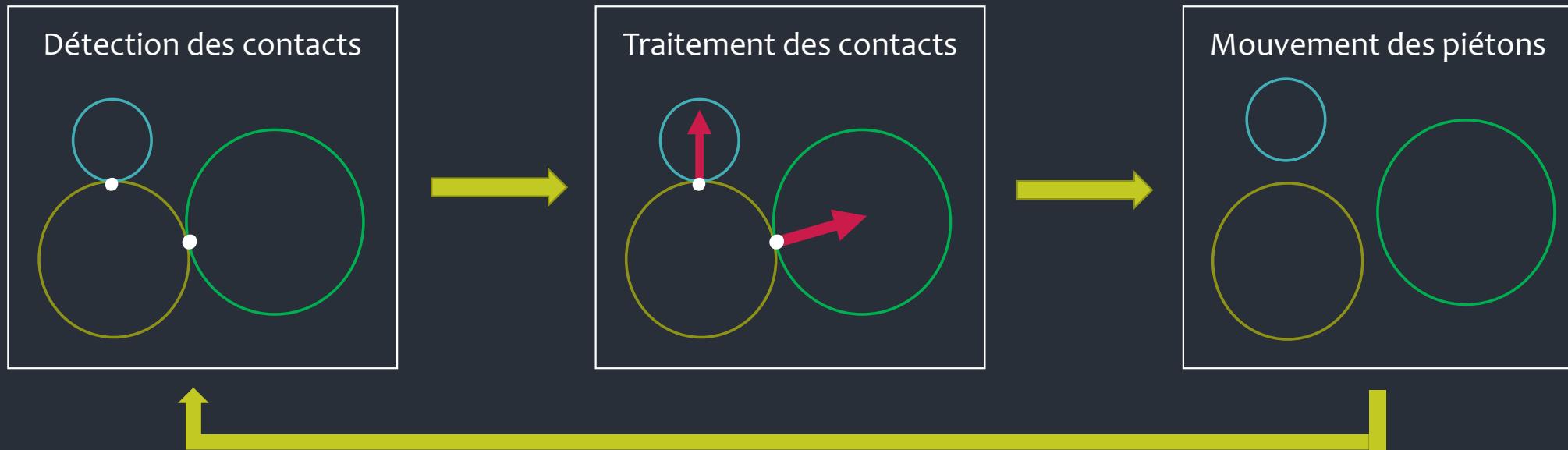
Modèle des forces sociales

□ Force de contact :

La force de contact modélise les interactions de répulsion du piéton avec son environnement, et prend en compte:

- ✓ Le contact piéton-piéton
- ✓ Le contact piéton-obstacle

La détermination de cette force se fait selon la procédure suivante :



Modèle des forces sociales

Dans un premier temps, on ne s'intéresse qu'aux forces de contact **piéton-piéton**. Les interactions **piéton-obstacle** se déterminent analogiquement.

1- Détection des contacts:

On définit la distance algébrique entre 2 piétons i et j par :

$$D_{ij} = |q_j - q_i| - (r_j + r_i)$$

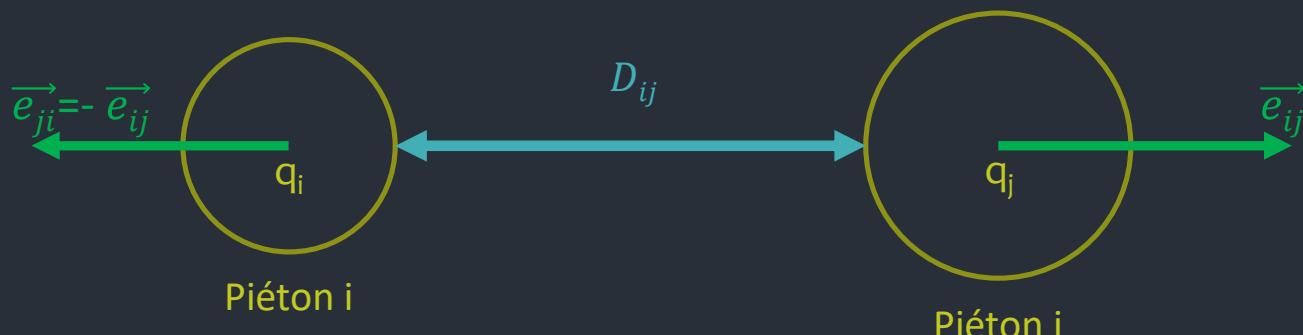
Avec :

q_i : les coordonnées de position du piéton i. $q_i = (x_i, y_i)$

r_i : le rayon du piéton i.

$$|q_j - q_i| = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

- ✓ Il y a contact entre les particules i et j lorsque $D_{ij} = 0$, et un chevauchement lorsque $D_{ij} < 0$



Modèle des forces sociales

2- Traitement des contacts:

Soit i et j l'indices de deux piétons dans une foule.

La force de contact de j appliquée sur i s'écrit sous la forme :

$$\overrightarrow{g_{j \rightarrow i}} = k \cdot \min(0, D_{ij}) \cdot \overrightarrow{e_{ij}}$$

Avec :

D_{ij} : la distance entre i et j

$\overrightarrow{e_{ij}}$: le vecteur directeur unitaire dirigé de i vers j

k : constante de raideur

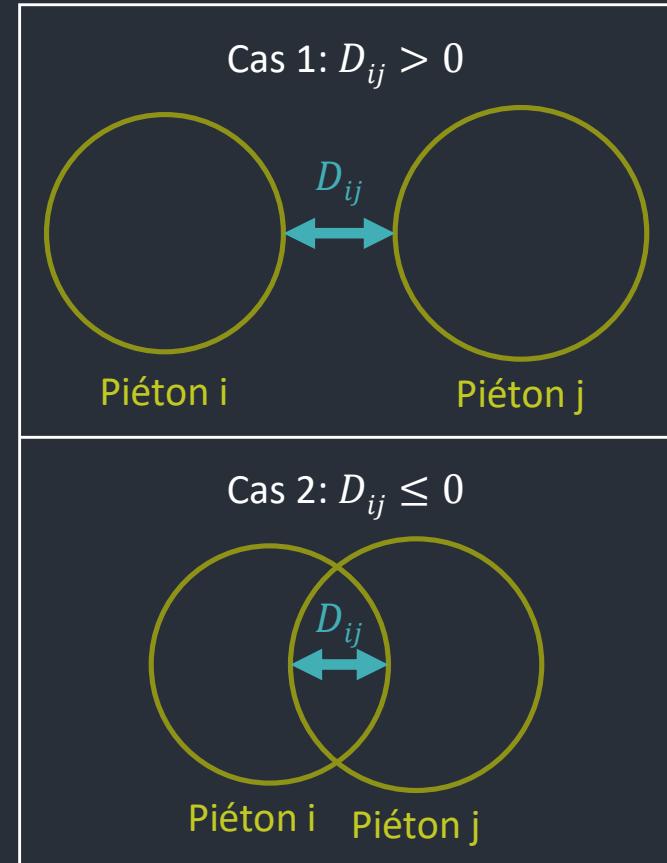
On prend $k = 1,2 \cdot 10^3$ (d'après Dirk Helbing)

Donc la force s'écrit :

$$\overrightarrow{g_{j \rightarrow i}} = \begin{cases} \vec{0} & \text{si } D_{ij} > 0 \\ k \cdot D_{ij} \cdot \overrightarrow{e_{ij}} & \text{si } D_{ij} \leq 0 \end{cases}$$

Et donc la force totale de contact piéton-piéton s'écrit :

$$\overrightarrow{g_i} = \sum_{\substack{j=1 \\ j \neq i}}^N \overrightarrow{g_{j \rightarrow i}}$$



Modèle des forces sociales

3- Mouvement des piétons:

Les piétons sont soumis à une multitude de forces. On peut donc déterminer leurs positions à chaque instant en appliquant le **Principe Fondamentale de la Dynamique (PFD)**

Soit i un piéton, il est soumis aux trois forces élaborées précédemment.

Appliquons le PFD sur le piéton i dans le référentiel terrestre considéré Galiléen:

$$m_i \cdot \ddot{q}_i(t) = f_i(t) + g_i(t) + k_i(t)$$

Avec:

m_i : la masse du piéton i

\ddot{q}_i : l'accélération du piéton i .

f_i, g_i, k_i : les trois forces motrice, de contact piéton-piéton, de contact piéton-obstacle

Par une subdivision du temps de travail $[0, T]$ à N intervalles $[t_n, t_{n+1}]$ de longueur $h=T/N$; on trouve :

$$\begin{cases} v_i^{n+1} = v_i^n + \frac{h}{m_i} (f_i^n + g_i^n + k_i^n) \\ q_i^{n+1} = q_i^n + h \cdot v_i^n \end{cases}$$

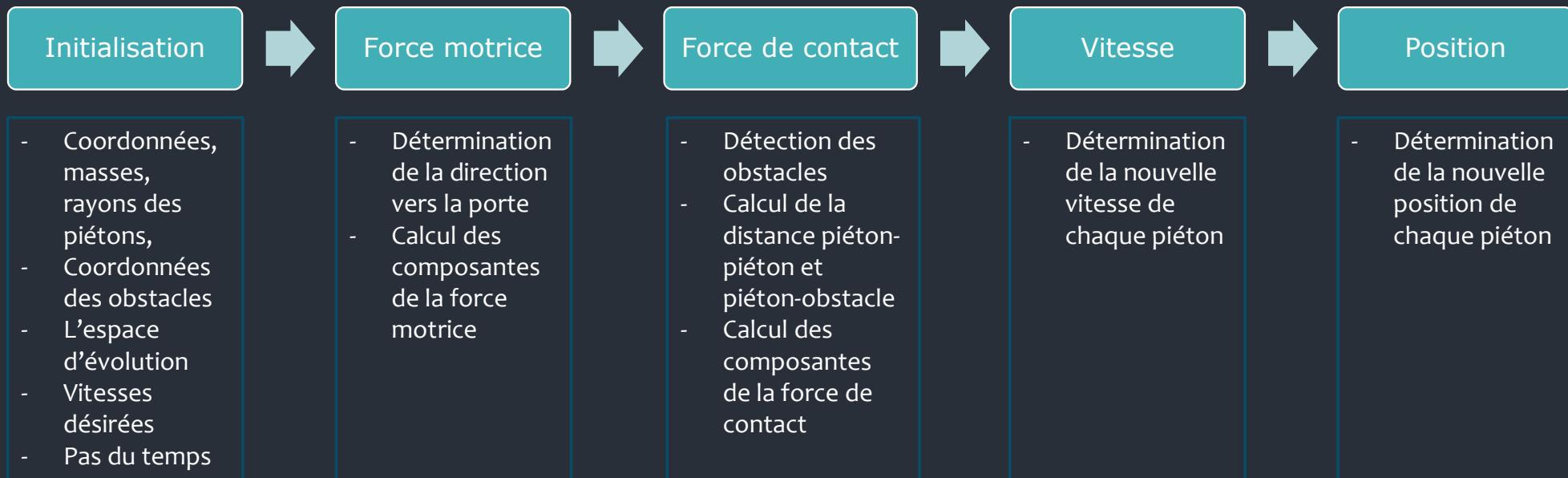
Avec :

$$v_i = \frac{dq_i}{dt}$$

5- Application du modèle établi

Simulation des situations de panique

Schéma illustrant les étapes du modèle discret du mouvement de la foule:



Dans les simulations, on prend :

$$\begin{aligned}
 50kg &\leq m_i \leq 100kg \\
 1,1\text{ m/s} &\leq v_i^0 \leq 1,48\text{ m/s} \\
 0,2m &\leq r_i \leq 0,25m
 \end{aligned}$$

Application 1 : Evacuation d'une salle vide

La simulation de l'évacuation de la salle vide :

- La salle est de dimensions (5m x 5m)
- Les piétons se trouvent initialement dans des positions aléatoires dans la salle

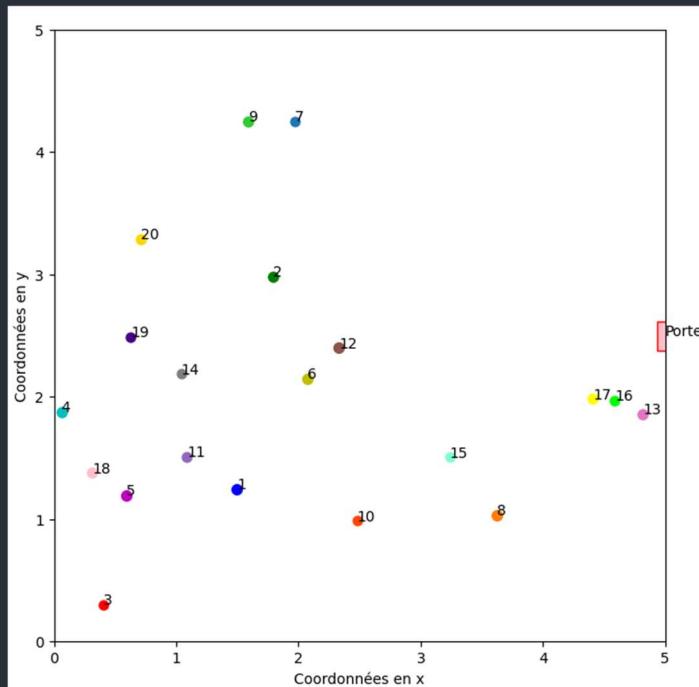


Figure 6 : Représentation coordonnées initiales aléatoires des piétons

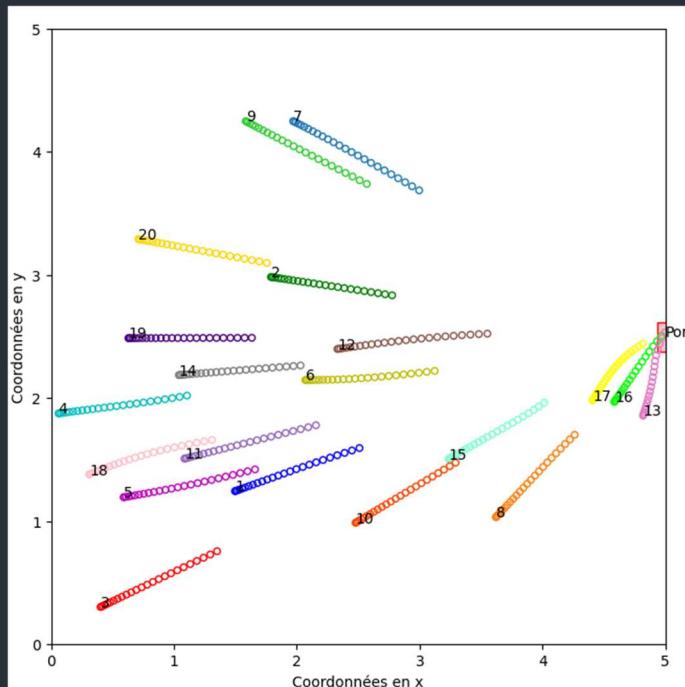


Figure 7 : Traçage du chemin des piéton jusqu'à l'instant $t=1,6s$

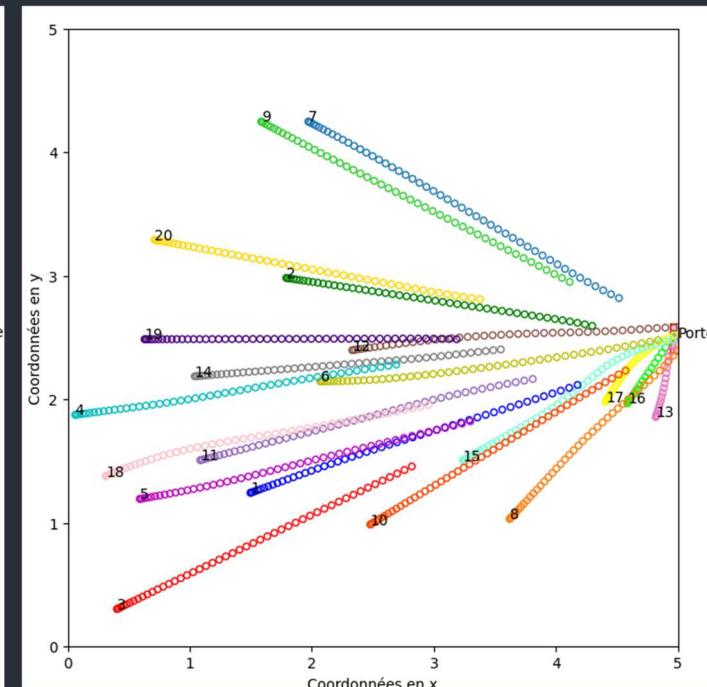


Figure 8 : Traçage du chemin des piéton jusqu'à l'instant $t=3,2s$

Application 1 : Evacuation d'une salle vide

Amélioration du modèle :

La force de réaction du mur sur un piéton i est donnée par :

k : la constante de raideur

Avec :

D : la distance entre le mur et le disque qui représente le piéton

\vec{e}_x : le vecteur directeur de l'axe des abscisses

$$\vec{k}_{j \rightarrow i} = \begin{cases} \vec{0} & \text{si } D > 0 \\ -k \cdot D \cdot \vec{e}_x & \text{si } D \leq 0 \end{cases}$$

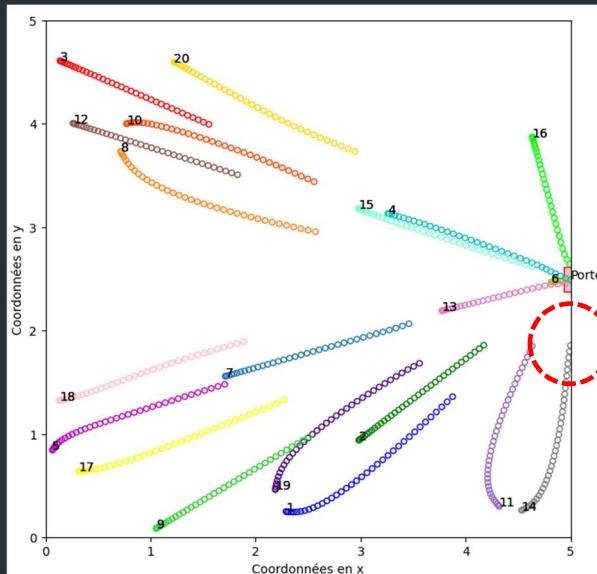


Figure 9 : Résultats obtenues sans l'utilisation de la réaction du mur

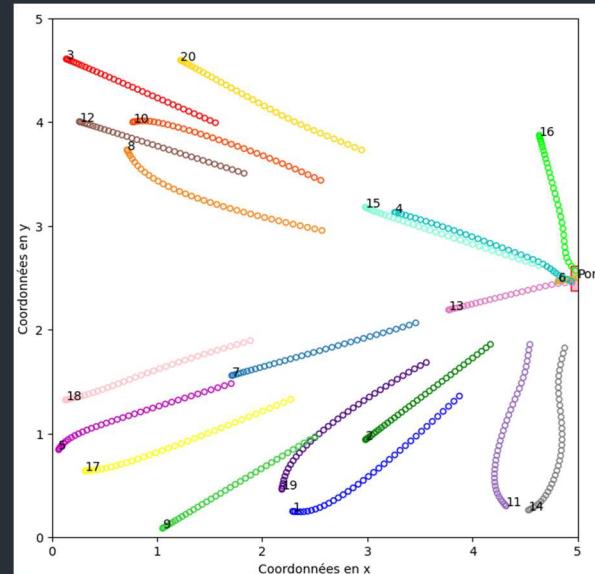


Figure 10 : Résultats obtenues avec l'utilisation de la réaction du mur

Application 1: Evacuation d'une salle vide

On définit Q le débit moyen des piétons traversant la porte

La régression linéaire des 50 simulations donne une droite affine de pente : $Q = 4,5$ piéton/s

Le tableau suivant présente une comparaison entre Q obtenu par simulations et celui obtenu par l'exercice réel :

	Débit moyen
Simulation	4,5 piéton/s
Exercice réel (M. Moussaid)	4,2 piéton/s

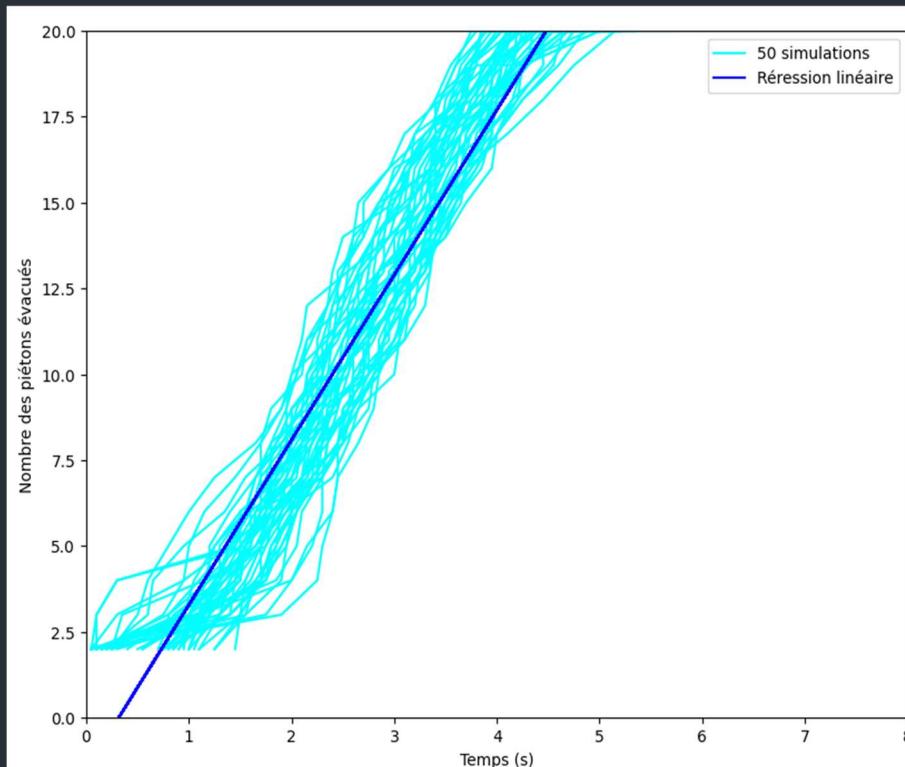


Figure 11 : Evacuation d'une salle vide – Résultats obtenus par 50 simulations et la régression linéaire des résultats

Application 2 : Evacuation d'une salle de classe

La simulation de l'évacuation de la salle de ma classe.

- La salle est de dimensions (7m x 8,4m)
- Les étudiant se trouvent initialement dans leurs tables qui sont des obstacles de dimensions (35cm x 50cm)

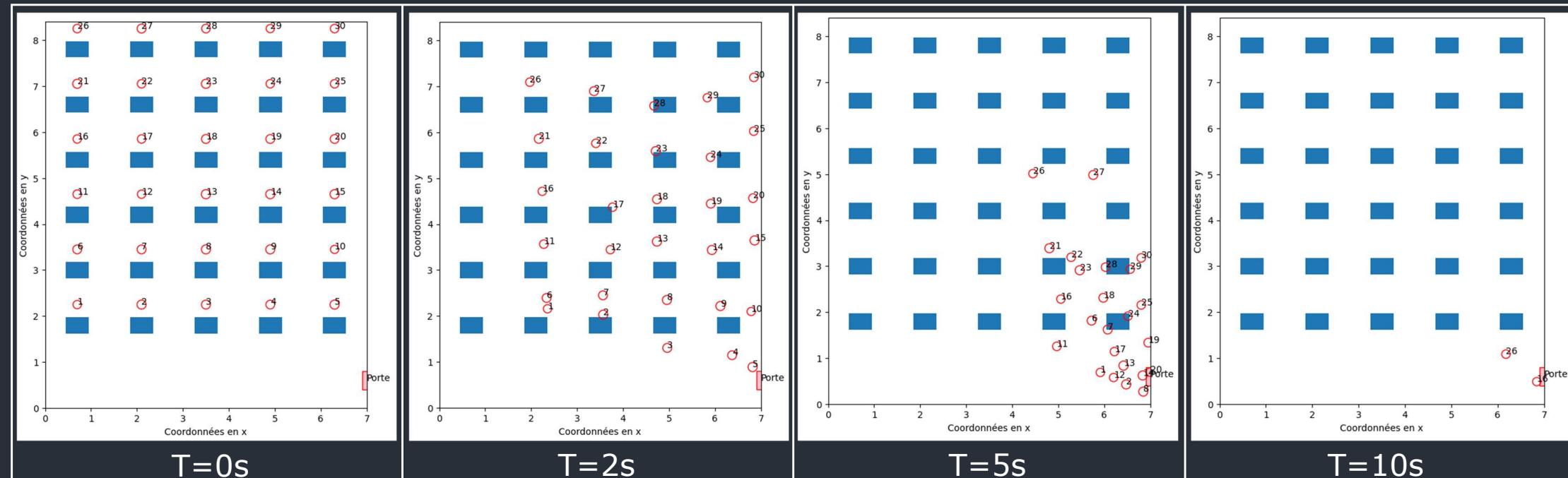


Figure 12: Evacuation d'une salle de classe - Exemple de progression d'une simulation numériques à différents instants

Application 2 : Evacuation d'une salle de classe

Comparaison entre les simulations et l'exercice réel:

Le tableau suivant présente une comparaison entre temps d'évacuation de la totalité des élèves obtenu par simulations et celui obtenu par l'exercice réel :

	Temps d'évacuation
Simulation	12,2 s
Exercice réel (D. Hilbing)	13 s

Application 2 : Evacuation d'une salle de classe

Influence de la valeur du pas du temps h :

En faisant 2 simulations avec des pas du temps différents, on obtient:

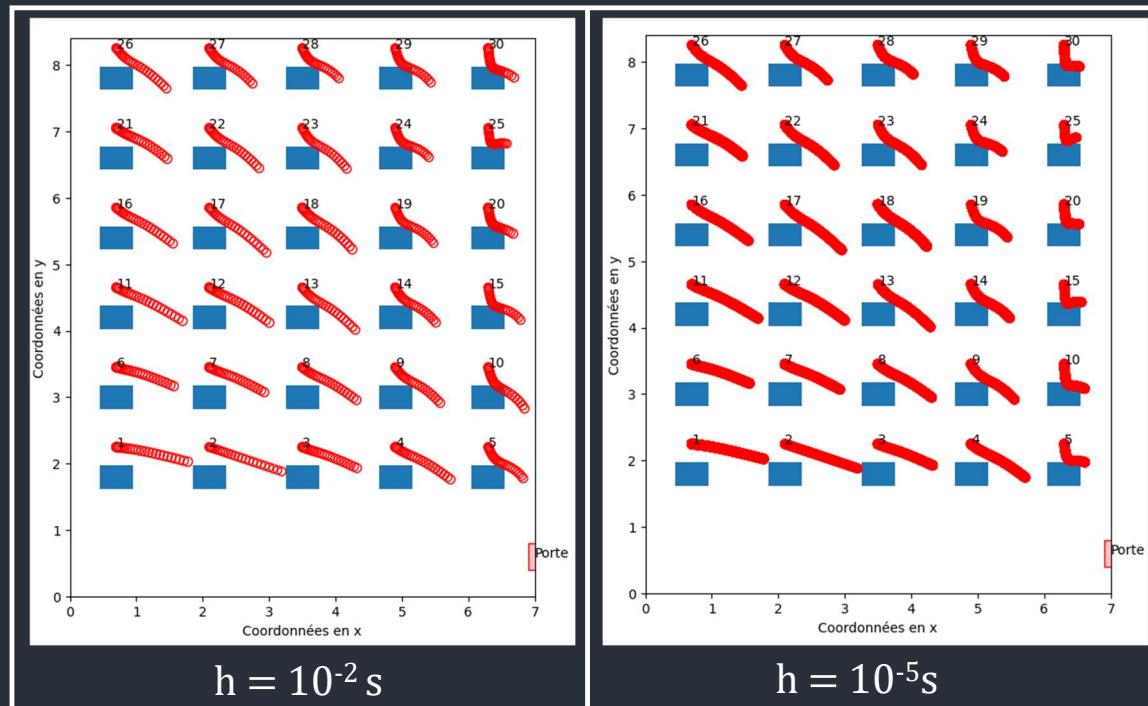


Figure 13 : Traçage du chemin des piétons jusqu'à $t=1,25\text{s}$ pour différentes valeurs du pas du temps h

Pas du temps	Temps d'évacuation de tous les élèves
$h=10^{-2} \text{ s}$	12,1 s
$h=10^{-5} \text{ s}$	11,6 s

Application 3 : L'effet 'Slower is faster'

L'observation des foules grande densité montre que:

- ✓ Si la vitesse souhaitée est supérieure à 1,5m/s, il y a blocage des individus.
- ✓ Si le nombre des piétons augmente, l'embouteillage s'augmente, et donc le temps d'évacuation augmente
- ✓ Le plus les piétons veulent sortir vite, le moins vite ils sortent

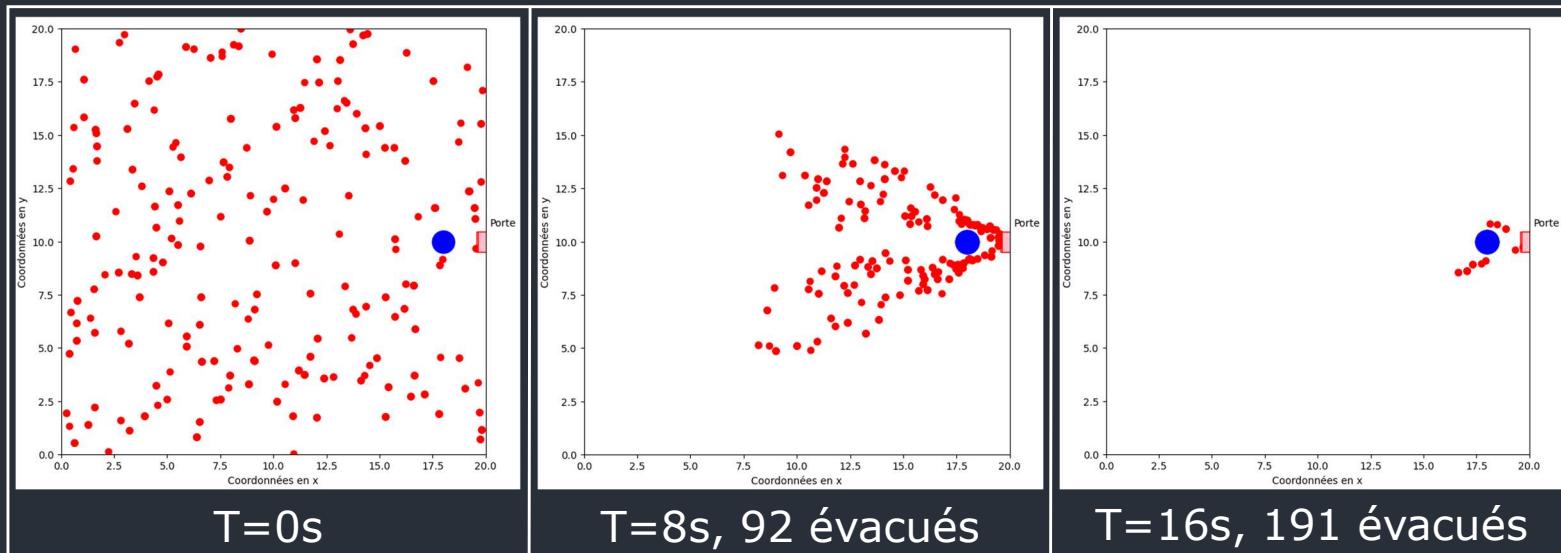


Figure 14 : Effet 'Slower is faster' - Exemple de progression d'une simulation numériques qui comporte un obstacle en amont de la sortie à différents instants

Application 3 : L'effet 'Slower is faster'

Comparaison entre le cas avec et sans obstacle en amont de la porte:

- La salle est de dimensions (20m x 20m)
- Les piétons se trouvent initialement dans des positions aléatoires dans la salle

Le tableau suivant présente une comparaison entre temps d'évacuation de la totalité des piétons dans le cas sans et avec l'obstacle :

	Temps d'évacuation
Sans obstacle	19,6 s
Avec l'obstacle	16,5s

6- Conclusion



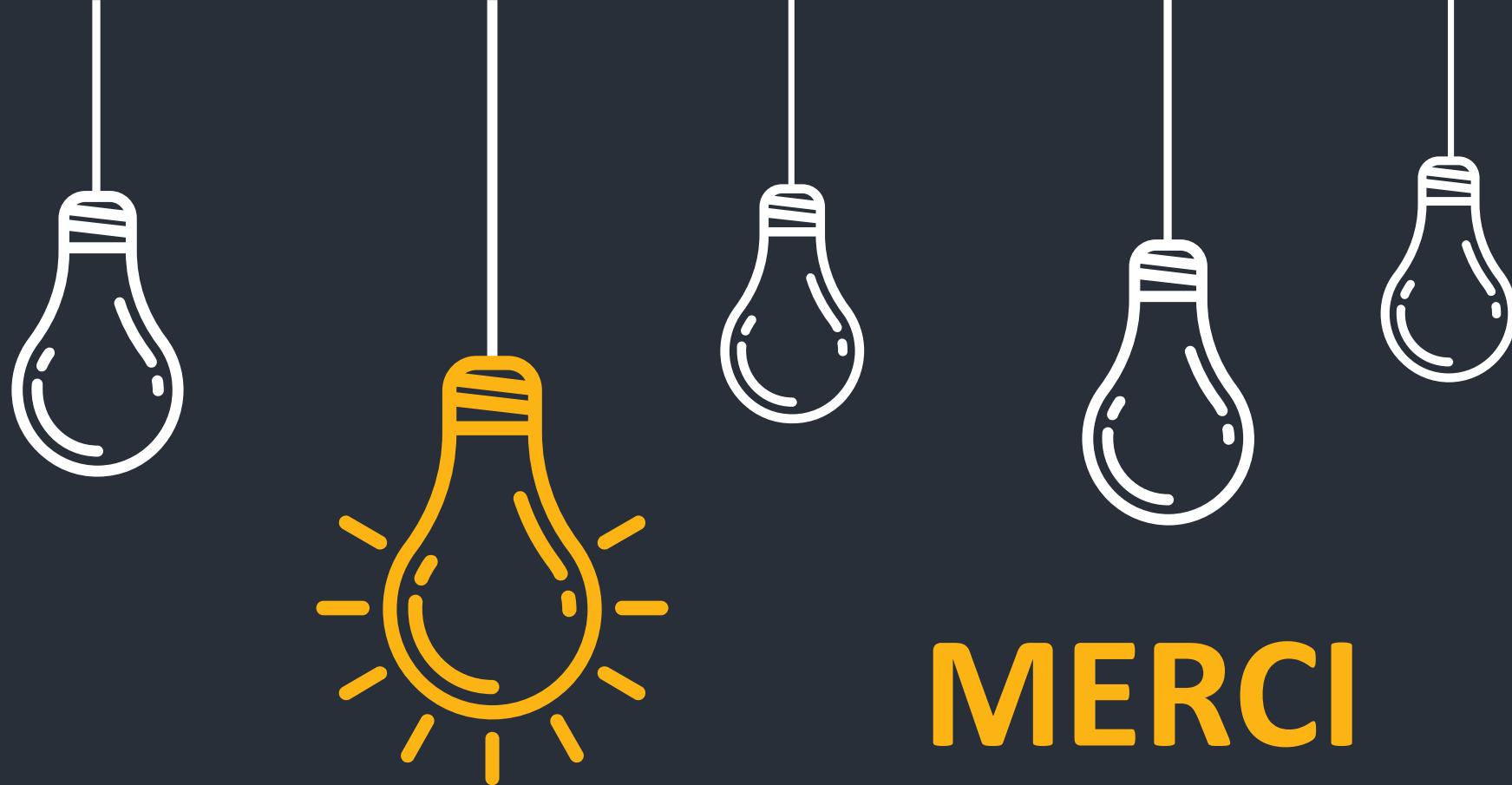
Réponse la problématique:

- On peut donc modéliser les foules de piétons par des points matériels représentés par des disques
- Les lois qui gèrent le mouvement des foules en situations de panique s'induisent en utilisant le PFD
- Le modèle des forces sociales permet l'obtention des résultats proches de la réalité



Un modèle qui permet la prévention dans:

- La construction des futures aménagements industriels
- Le calibrage des positions, et des dimensions des voies d'évacuations
- L'estimation du temps nécessaire pour l'évacuation
- La détermination des positions d'obstacles pouvant optimiser l'évacuation d'urgence



MERCI
pour votre
attention

Annexe

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import math
4 import random
5 from sklearn.linear_model import LinearRegression
6 import pickle
7 from matplotlib.patches import Rectangle
8
9 #initialisation
10 nombrePietons=20 ; largeurPiece=5 ; longeurPiece=5 ; k=1.2*(10**3)
11 h=10**(-2) #pas du temps
12 T=0.5 #temps de relaxation
13 listePortes=[[largeurPiece,longeurPiece/2]] #en cas d'existance de plusieurs portes
14 Vn=np.zeros((nombrePietons,2)) #vitesse initiale
15
16
17 def initialisation():
18     global coordonnees,vitesseSouhaitee,masse,rayon
19
20     #génération des coordonnées aléatoires des piétons
21     coordonnees=np.zeros((nombrePietons,2))
22     for i in range(nombrePietons):
23         coordonnees[i,0]=(random.uniform(0, largeurPiece))
24         coordonnees[i,1]=(random.uniform(0, longeurPiece))
25
26     #génération des vitesses souhaitées aléatoires entre 1.1m/s et 1.48 m/s
27     vitesseSouhaitee=np.zeros(nombrePietons)
28     for i in range(nombrePietons):
29         vitesseSouhaitee[i]=random.uniform(1.1,1.48)
30
31     #génération des masses aléatoires entre 50kg et 100kg
32     masse=np.random.randint(low=50, high=100, size=nombrePietons)
33
34     #choix des rayons convenables des pietons entre 0.2m et 0.25m
35     rayon=np.array( [ 0.001*masse[i]+0.15 for i in range(nombrePietons) ] )
```

```
37 initialisation()
38
39 #listes des couleurs choisi pour représenter chaque piéton
40 t='tab:'
41 colors = ['b','g','r','c','m','y',t+'blue',t+'orange','limegreen','orangered',t+'purple',t+'brown',t+'pink',t+'grey','aquamarine','lime','yellow','pink','indigo','gold']
42
43
44 #tracage des données initiales
45 def environnement():
46     fig, ax = plt.subplots()
47     ax.plot()
48     plt.axis('square')
49     plt.xlim([0, largeurPiece]); plt.ylim([0, longeurPiece])
50     plt.xlabel('Coordonnées en x') ; plt.ylabel('Coordonnées en y')
51     plt.scatter(listePortes[0][1]+0.05,listePortes[0][1],color= 'pink',marker='s',edgecolors='r',s=400, label='Porte')
52     plt.text(listePortes[0][0],listePortes[0][1],"Porte")
53
54 environnement()
55 for N in range(nombrePietons):
56     plt.scatter(coordonnees[N,0],coordonnees[N,1], color=colors[N],s=rayon[N]**200)
57     plt.text(coordonnees[N,0],coordonnees[N,1], str(N+1))
58
59
60 #définition des fonctions:
61
62 #calcul de la distance entre le piéton et la porte
63 def distanceToDoor(numeroPieton,coordsPorte):
64     return math.sqrt( (coordsPorte[0]-coordonnees[numeroPieton][0])**2 +(coordsPorte[1]-coordonnees[numeroPieton][1])**2 )
65
66 #determination de la porte la plus proche
67 def closestDoor(numeroPieton,listePorte):
68     distances = []
69     for p in range(0,len(listePorte)):
70         distances.append(distanceToDoor(numeroPieton,listePorte[p]))
71     return distances.index(min(distances))
```

```

73 #calcul de la direction souhaitée d'un piéton (vers la plus proche porte)
74 def direction(numeroPieton):
75     porte=closestDoor(numeroPieton,listePortes)
76     X=listePortes[porte][0]- coordonnees[numeroPieton][0]
77     Y=listePortes[porte][1] -coordonnees[numeroPieton][1]
78     return np.array([X/distanceToDoor(numeroPieton,listePortes[porte]),Y/distanceToDoor(numeroPieton,listePortes[porte])])
79
80 #calcul de la force motrice F
81 def calculF(numeroPieton,Vn):
82     dir=direction(numeroPieton)
83     return (masse[numeroPieton]*(vitesseSouhaitee[numeroPieton]*dir-Vn[numeroPieton]))/T
84
85 #calcul de la distance entre deux piétons
86 def distanceEntre(i,j,coordonnees):
87     return math.sqrt((coordonnees[i][0]-coordonnees[j][0])**2+(coordonnees[i][1]-coordonnees[j][1])**2)
88
89 #détection des piétons en contact avec un piéton
90 def detectionContact(i,coordonnees):
91     piétonsTouchés=[]
92     for j in range(0,nombrePietons): #j pour les autres piétons
93         if j != i :
94             Dij = distanceEntre(i,j,coordonnees) - (rayon[i]+rayon[j])
95             if Dij <= 0 :
96                 piétonsTouchés.append(j)
97     return piétonsTouchés
98
99 #calcul de la force piéton-piéton G
100 def calculG(numeroPieton,coordonnees):
101     piétonsTouchés=detectionContact(numeroPieton,coordonnees)
102     G=np.zeros(2)
103     for j in piétonsTouchés:
104         Dij = distanceEntre(numeroPieton,j,coordonnees) - (rayon[numeroPieton]+rayon[j])
105         G=G+k*Dij*((coordonnees[j]-coordonnees[numeroPieton])/distanceEntre(j,numeroPieton,coordonnees))
106     return G

```

```
108 #calcul de la force pieton-mur de la porte K
109 def calculK(numeroPieton, coordonnees):
110     porte=closestDoor(numeroPieton, listePortes)
111     K=np.zeros(2)
112     if coordonnees[numeroPieton,1] < listePortes[porte][1]-0.2 or coordonnees[numeroPieton,1] > listePortes[porte][1]+0.2:
113         D=largeurPiece-coordonnees[numeroPieton,0]-rayon[numeroPieton]
114         if D <= 0 :
115             K[0]=K[0]+k*D
116     return K
117
118 #calcul de la vitesse d'un piéton à tout instant
119 def calculVitesse(numeroPieton, Vn, coordonnees):
120     return Vn[numeroPieton]+(h/masse[numeroPieton]) * (calculF(numeroPieton,Vn)+calculG(numeroPieton,coordonnees)+calculK(numeroPieton,coordonnees))
121
122 #Calcul de la position d'un piéton à tout instant
123 def calculPosition (numeroPieton,Vn, coordonnees):
124     return coordonnees[numeroPieton] + h*Vn[numeroPieton]
125
126
127 #tracage des trajets des piétons dans une salle vide (figures 6,7,8,9 et 10 en ajoutant la force k)
128 environnement()
129 for N in range(nombrePietons):
130     plt.text(coordonnees[N,0],coordonnees[N,1], str(N+1))
131
132 i=0
133 while min(coordonnees[:,0])<listePortes[0][0]: #pour s'assurer que tous les piétons ont évacués
134     for N in range(0,nombrePietons):
135         if coordonnees[N,0]<listePortes[0][0]:
136             plt.scatter(coordonnees[N,0],coordonnees[N,1],facecolors='none', edgecolors=colors[N],s=20)
137             Vn[N]=calculVitesse(N,Vn,coordonnees)
138             coordonnees[N]=calculPosition(N,Vn,coordonnees)
139         else:
140             coordonnees[N]=np.array([1000,1000]) #pour éviter de prendre en compte les interactions des piétons qui sont sortis
141 i=i+1 #pour determiner le temps d'évacuation: on multiplie i final par h le pas du mouvement.
142 if i == 25:
143     temp=pickle.dumps(ax)
144     plt.show()
145     ax=pickle.loads(temp)
146 if i == 50:
147     break
148 plt.show()
```

```
151 #tracage des résultats de 50 simulations et la regression linéaire (figure 11)
152
153 initialisation()
154 fig, ax = plt.subplots()
155 plt.xlim([0, 8]); plt.ylim([0, 20])
156 plt.yticks(np.arange(0,21,2))
157 plt.xlabel('Temps (s)') ; plt.ylabel('Nombre des piétons évacués')
158
159 #tracage des 50 simulations
160 X,Y=[],[]
161 for j in range(50):
162     initialisation()
163     i=0 ; p=0 ; Temps=[] ; Nombre=[]
164     while min(coordonnees[:,0])<listePortes[0][0]:
165         for N in range(0,nombrePietons):
166             if coordonnees[N,0]<listePortes[0][0]:
167                 Vn[N]=calculVitesse(N,Vn,coordonnees)
168                 coordonnees[N]=calculPosition(N,Vn,coordonnees)
169             else:
170                 if coordonnees[N][0] != 1000 :
171                     p+=1
172                     Temps.append(i*h)
173                     Nombre.append(p+1)
174                     X.append(i*h)
175                     Y.append(p+1)
176                     coordonnees[N]=np.array([1000,1000])
177             i+=1
178     Temps.append(i*h)
179     Nombre.append(p+1)
180     X.append(i*h)
181     Y.append(p+1)
182     if j == 0 :
183         plt.plot(Temps,Nombre,color='cyan',label='50 simulations')
184     else :
185         plt.plot(Temps,Nombre,color='cyan')
```

```
187 #tracage de la régression linéaire
188 regr=LinearRegression()
189 X=np.array(X).reshape((-1,1))
190 Y=np.array(Y)
191 regr.fit(X,Y)
192 pente=regr.coef_
193 print(pente) #le coefficient de la pente de la droite
194 Z=regr.predict(X)
195 plt.plot(X,Z,color='b',label='Régression linéaire')
196 plt.legend()
197 plt.show()
198
199
200 #tracage de la progression des étudiants dans la salle d'une classe (figures 12,13)
201
202 coord1=[]
203 for i in np.arange(0,6.1,1.2):
204     coord1=coord1+[(0.7+j,1.8+i)for j in np.arange(0,5.9,1.4)]
205
206 #coordonnées des objets dans la classe
207 coordObjets=np.zeros((nombrePietons,2))
208 for i in range(nombrePietons):
209     coordObjets[i,0]=coord1[i][0]
210     coordObjets[i,1]=coord1[i][1]
211
212 #coordonnées initiales des etudiants dans la classe
213 coordonnees=np.zeros((nombrePietons,2))
214 for i in range(nombrePietons):
215     coordonnees[i,0]=coord1[i][0]
216     coordonnees[i,1]=coord1[i][1]+0.45
217
218 #placement des objets (les tables)
219 L=[] #liste des coordonnees des objets
220 for i in np.arange(0,6.1,1.2):
221     L=L+[(0.45+j,1.625+i)for j in np.arange(0,5.9,1.4)]
```

```

223 def environnement():
224     ax.plot()
225     plt.axis('square')
226     plt.xlim([0, largeurPiece]); plt.ylim([0, longeurPiece])
227     plt.xlabel('Coordonnées en x') ; plt.ylabel('Coordonnées en y')
228     for i in L:
229         ax.add_patch(Rectangle(i,0.5,0.35))
230     plt.scatter(listePortes[0][0]+0.1,listePortes[0][1],color= 'pink',marker='s',edgecolors='r',s=400, label='Porte')
231     plt.text(listePortes[0][0],listePortes[0][1],"Porte")
232
233 #définition des fonctions
234
235 #calcul de la distance entre le piéton et l'objet
236 def distanceEntreObj(i,j,coordonnees,coordObjets):
237     return math.sqrt((coordonnees[i][0]-coordObjets[j][0])**2+(coordonnees[i][1]-coordObjets[j][1])**2)
238
239 #déttection des objets en contact avec un piéton
240 def detectionContactObjets(i,coordonnees,coordObjets):
241     objetsTouches=[]
242     for j in range(0,nombrePietons): #j pour les objets (car nombrePietons=nombreObjets)
243         Dij = distanceEntreObj(i,j,coordonnees,coordObjets) - 2*rayon[i]
244         if Dij <= 0 :
245             objetsTouches.append(j)
246     return objetsTouches
247
248 #calcul de la force pieton-objet L
249 def calculL(numeroPieton,coordonnees,coordObjets):
250     objetsTouches=detectionContactObjets(numeroPieton,coordonnees,coordObjets)
251     L=np.zeros(2)
252     for j in objetsTouches:
253         Dij = distanceEntreObj(numeroPieton,j,coordonnees,coordObjets) - 2*rayon[numeroPieton]
254         L=L+k*Dij*((coordObjets[j]-coordonnees[numeroPieton])/distanceEntreObj(numeroPieton,j,coordonnees,coordObjets))
255     return L
256
257 #calcul de la vitesse d'un piéton à tout instant (en ajoutant la force L)
258 def calculVitesse(numPieton,Vn,coordonnees):
259     return Vn[numPieton]+(h/masse[numPieton])*(calculF(numPieton,Vn)+calculG(numPieton,coordonnees)+calculK(numPieton,coordonnees)+calculL(numPieton,coordonnees,coordObjets))

```

```
262 #tracage des positions des étudiants
263 def tracage():
264     for N in range(0,nombrePietons):
265         plt.scatter(coordonnees[N][0],coordonnees[N][1],facecolors='none',edgecolors='r',s=rayon[N]**200)
266         plt.text(coordonnees[N][0],coordonnees[N][1], str(N+1))
267
268 fig, ax = plt.subplots()
269 environnement()
270 tracage() #t=0s
271
272 i=0
273 while min(coordonnees[:,0])<listePortes[0][0]:
274     for N in range(0,nombrePietons):
275         if coordonnees[N][0]<listePortes[0][0]:
276             Vn[N]=calculVitesse(N,Vn,coordonnees)
277             coordonnees[N]=calculPosition(N,Vn,coordonnees)
278         else:
279             coordonnees[N]=np.array([1000,1000])
280     i=i+1
281     if i == 200:
282         temp=pickle.dumps(ax)
283         tracage() #t=i*h donc t=2s
284         plt.show()
285         ax=pickle.loads(temp)
286     if i == 500:
287         temp=pickle.dumps(ax)
288         tracage() #t=5s
289         plt.show()
290         ax=pickle.loads(temp)
291     if i==1000 :
292         break
293 tracage() #t=10s
294 plt.show()
```

```
297 #tracage de l'effet 'Slower is faster' (figure 14)
298 nombrePietons=200 ; largeurPiece=20 ; longeurPiece=20
299 environnement()
300 initialisation()
301 tracage() #t=0s
302 p=0 #compteur des piétons qui ont sorti
303 i=0
304 while min(coordonnees[:,0])<listePortes[0][0]:
305     for N in range(0,nombrePietons):
306         if coordonnees[N,0]<listePortes[0][0]:
307             Vn[N]=calculVitesse(N,Vn,coordonnees)
308             coordonnees[N]=calculPosition(N,Vn,coordonnees)
309         else:
310             if coordonnees[N][0] != 1000 : p+=1 #p nombre des piétons évacués
311             coordonnees[N]=np.array([1000,1000])
312     i=i+1
313     if i == 800:
314         temp=pickle.dumps(ax)
315         tracage() #t=8s
316         plt.show()
317         ax=pickle.loads(temp)
318     if i == 1600:
319         break
320 tracage() #t=16s
321 print("le nombre des piétons évacués est :",p)
322 plt.show()
```