# MODELISATION DU MOUVEMENT DE LA FOULE

GHAOUI FATIMA EZZAHRAE
TIPE 2021

# SOMMAIRE


01 INTRODUCTION

02 OBSERVATIONS DES PIETONS

03 MODELE DES FORCES SOCIALES

04 APPLICATIONS DU MODELE

05 CONCLUSION

# 1. INTRODUCTION:

**Figure 1: Une fête qui se termine en drame à Phnom Penh .**

**Figure 2:La bousculade qui a fait au moins 45 morts en Israël.**

**Figure 3: un agent de sécurité.**

**Figure 4: évacuation d'urgence.**

# 2. OBSERVATIONS DES PIETONS :

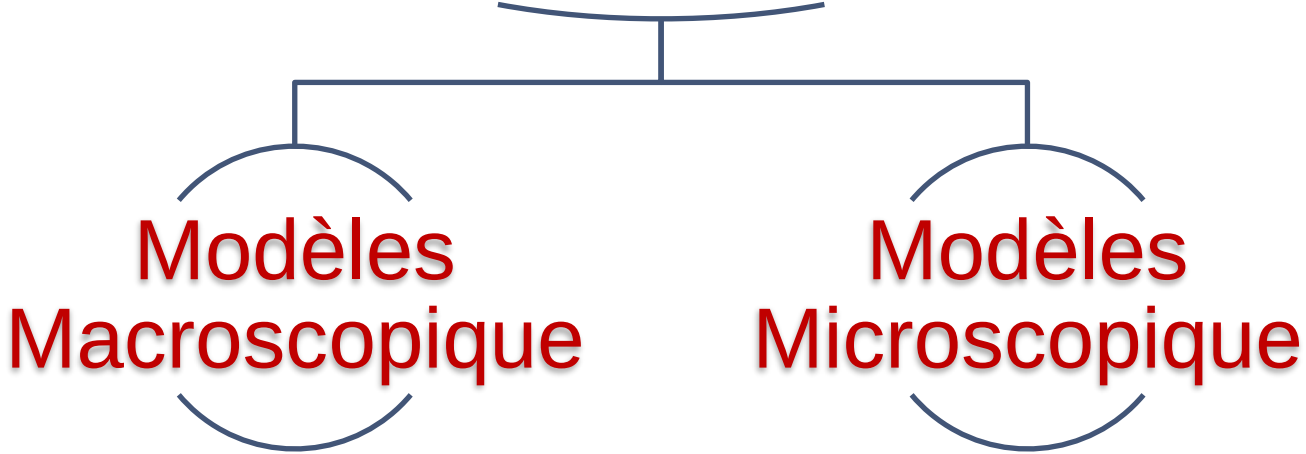
Chaque piéton admet :

- ✓ Une vitesse souhaitée, c'est sa vitesse en absence d'interaction avec les autres, elle est de l'ordre de 1.34 m/s avec un écart-type de 0.26 m/s.
- ✓ Une direction souhaité rectifiée incessamment selon les interactions piéton-piéton et piéton-obstacle.

En situation d'urgence, la vitesse des piétons augmente considérablement.

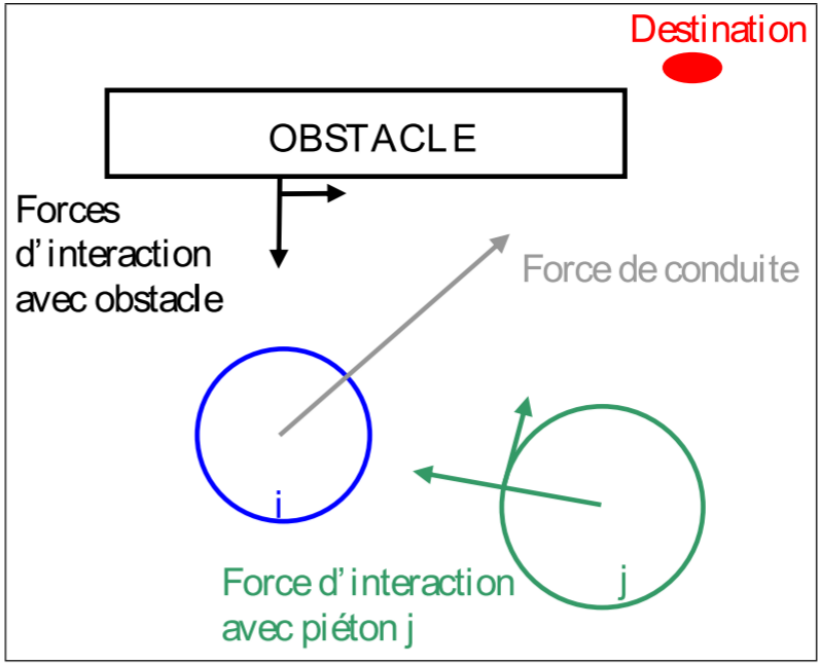
| AUTEUR | Valeur de vitesse souhaitée relevée |
|---|---|
| Dirk Helbing (2000) | 1.3 m/s |
| L.F.Henderson (1971) | 1.34 m/s |
| R.W.Bonhannon (1997) | Entre 1.27 m/s et 1.46 m/s |

# Modèles de la foule



# 3. MODELE DES FORCES SOCIALES :

- ✓ C’est un modèle discret 2D qui permet de gérer le mouvement de chaque piéton.
- ✓ Piétons identifiés a des disques de rayon $r_i$ masse $m_i$.
- ✓ Ces disques sont soumis a 2 forces: une force d’ accélération (motrice) et une force de contact.



# Force d'accélération motrice

Cette force exprime la volanté du piéton et assure son déplacement vers sa destination souhaitée.

$$\mathbf{f_i^a(t) = M_i \frac{||V_{si}||e_{si} - V_i(t)}{T_i}}$$

$$\mathbf{e_{si} = \frac{V_{si}}{||V_{si}||}}$$

Avec:

$M_i$ : la masse du piéton i

$e_{si}$: sa direction souhaitée

$V_i(t)$ :sa vitesse réelle

$V_{si}$ :sa vitesse souhaitée

$T_i$ : temps de relaxation (temps dans lequel il retrouve sa vitesse désirée après contact)

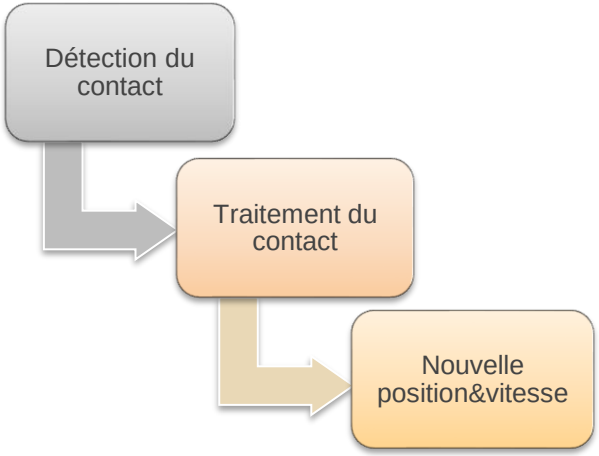
Dirk Helbing estime $T_i = 0.5\ s$

# Force de contact:

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

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

La détermination de cette force nécessite 3 étapes:



# Détection du contact

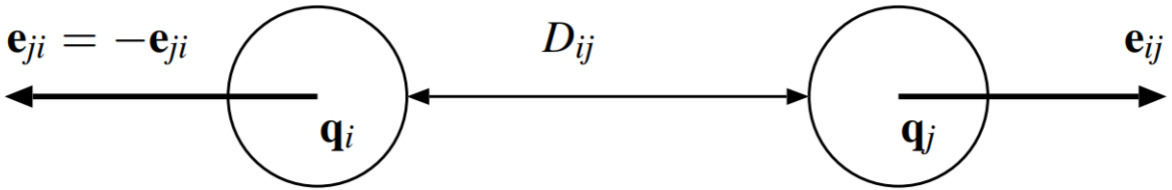
On considère uniquement les interactions piéton-piéton ,les interactions piéton-obstacle sont déterminées analogiquement.
On définit la distance entre 2 piétons i et j par:

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

$$|q_i - q_j| = \sqrt{|q_i^x - q_j^x|^2 + |q_i^y - q_j^y|^2}$$

Avec:

$q_i$: **position de l'agent i**
$r_i$: **rayon du disque de l'agent i**
Le contact est détecté entre les piétons i et j si $D_{ij} \leq 0$



# Traitement du contact

Le traitement du contact se fait par le biais d'une force répulsive $g_{ij}(t)$: elle représente l'interaction piéton-piéton entre 2 agents i et j:

$$g_{ij}(t)=K\ \min(0,D_{ij}(t))e_{ij}(t)$$

$$e_{ij}(t)=\frac{q_j-q_i}{|q_j-q_i|} \text{ et } K=1.2x10^5\ kg/S^2$$

Avec :

$e_{ij}$ : vecteur directeur unitaire de i ⇢j

K: constante de raideur

Le i -ème piéton est donc soumis a une force de contact gi avec les N-1 autres agents tel que:

$$g_i(t)=\sum_{\substack{j=1\\ j\neq i}}^{N} g_{ij}(t)$$

# Nouvelle position&vitesse

Pour un système de N agents, la rotation des disques est négligée, la positon et la vitesse réelle d'un agent i sont données par le système suivant :

$$\begin{cases} M_i \dot{V}_i(t) = f_i^a(t) + g_i(t) \\ \dot{q}_i(t) = V_i(t) \end{cases}$$

$q_i = (q_i^x, q_i^y) \in R^2$ : position de l'agent i a l'instant t .
$V_i = (V_i^x, V_i^y) \in R^2$ : vitesse de l'agent i a l'instant t.

Time-stepping : l'intervalle de temps $[0, T]$ sera divisé en N intervalles $[t^n, t^{n+1}]$ de longueur $h = T/N$.

Le système devient à l'instant $t^n$: $\begin{cases} M_i \dot{V}_i^n = f_i^{an} + g_i^a \\ \dot{q}_i^n = V_i^n \end{cases}$

À l'instant $t^{n+1}$: $\begin{cases} V_i^{n+1} = V_i^n + \frac{h}{M_i}(f_{ai}^n + g_i^n) \\ q_i^{n+1} = q_i^n + hV_i^n \end{cases}$

# 4.APPLICATION DU MODELE:

## SIMULATION DE L'EVACUATION D'UNE SALLE.

SHEMA DE LA SIMULATION

INITIALISATION → FORCE MOTRICE → FORCE DE CONTACT → VITESSE → POSITION

- ✓ Caractéristiques des piétons
- ✓ Obstacles
- ✓ Espace étudié, portes.

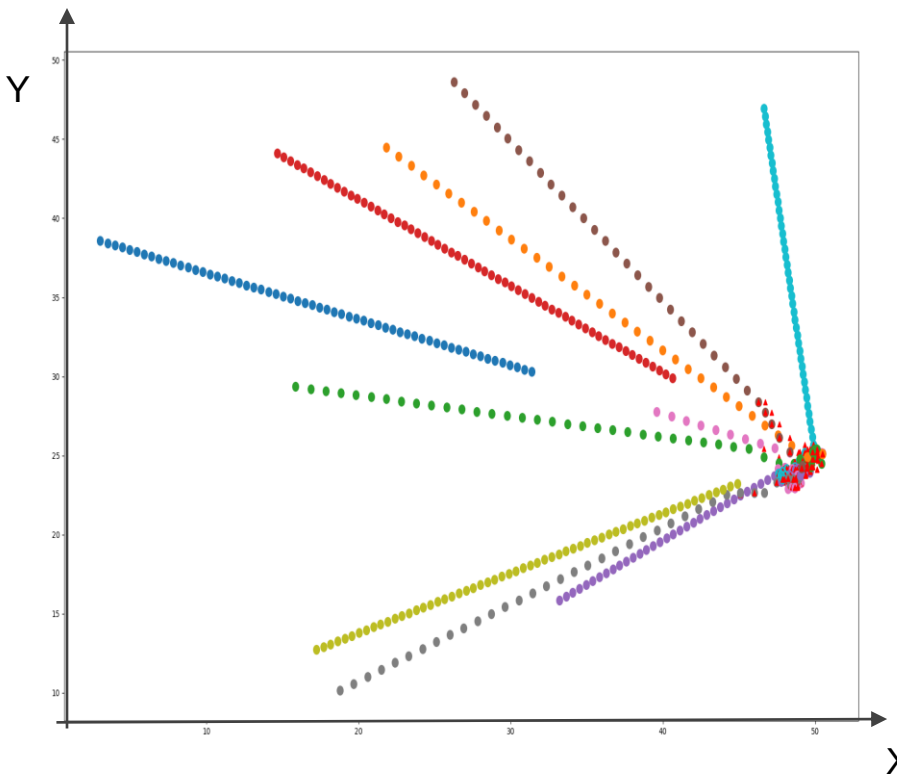
Calcul des composantes de le force $f^a$ du piéton i.

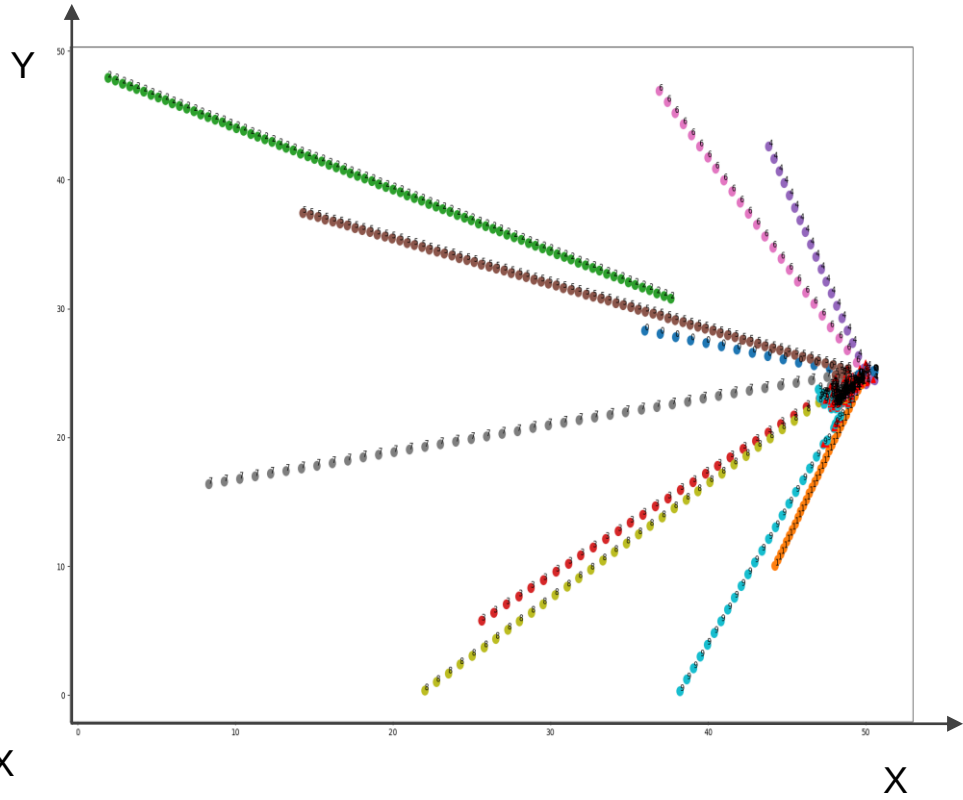
Calcul des composantes de la force de contact g.

Détermination de la nouvelle vitesse $V_{i.}$

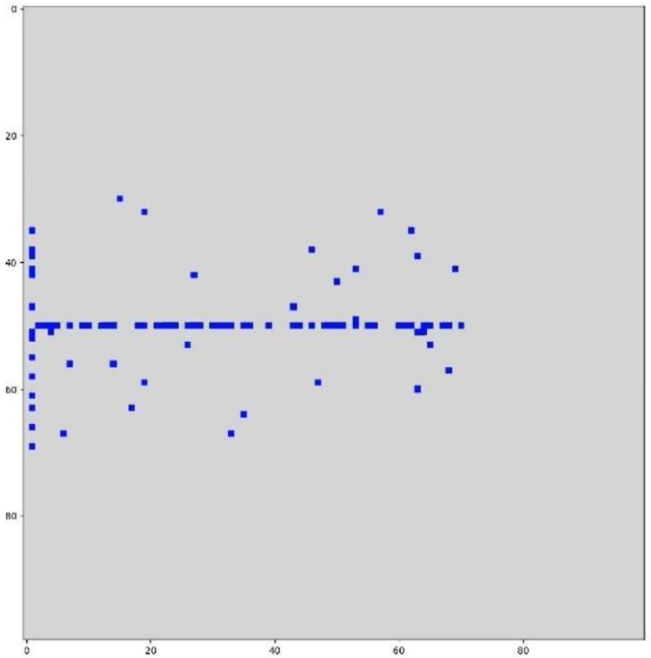
Détermination de la nouvelle position $q_{i.}$

## ➢ Exemple 1:

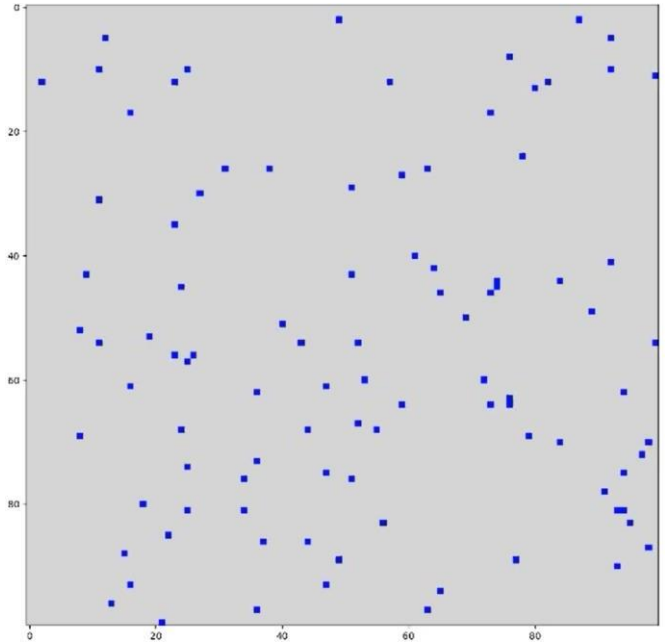



Figures 1&2 : simulations d'évacuation d'une salle pour 10 agents.



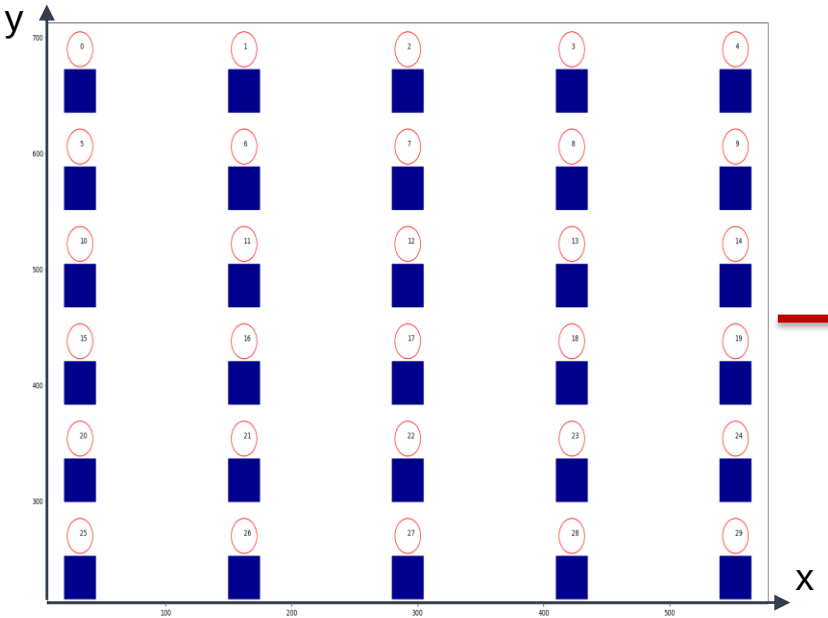

A t=3s



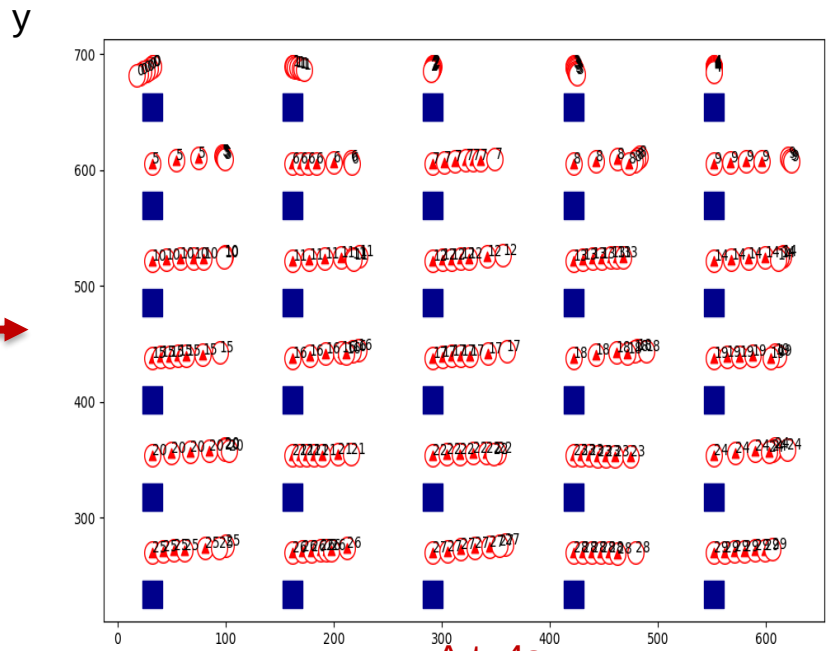

Figures 3&4 : images d'animation de l'évacuation d'une salle

# ➢ Exemple 2:

- Evacuation d'une salle de classe de  dimensions (6 m x 5 m) contenant 30 personnes uniformément distribuées avec 30 obstacles(tables).
- Introduction de la force de contact $K_{ij}$ qui modélise les interactions piéton-obstacle.
- La population étudiée est imposée :masse(min=45, max=100).

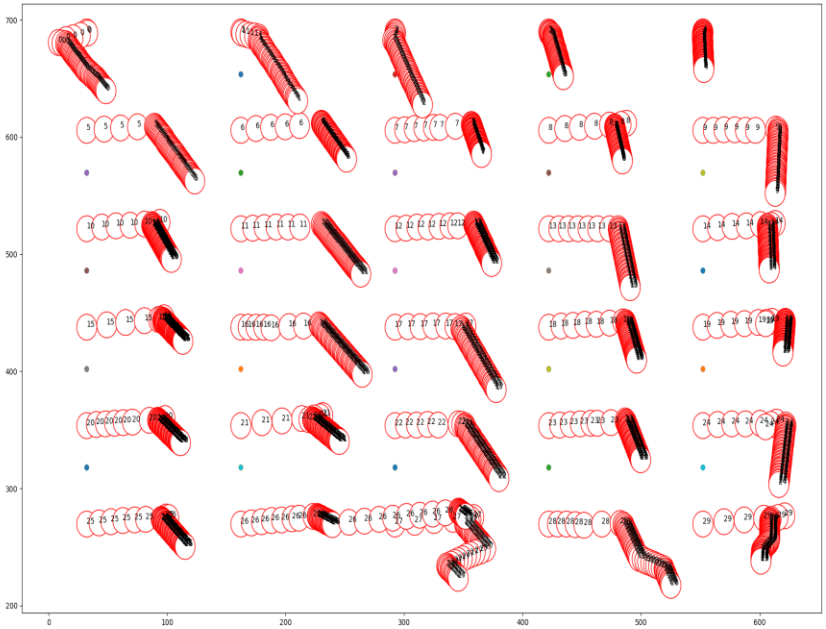



A t=0s

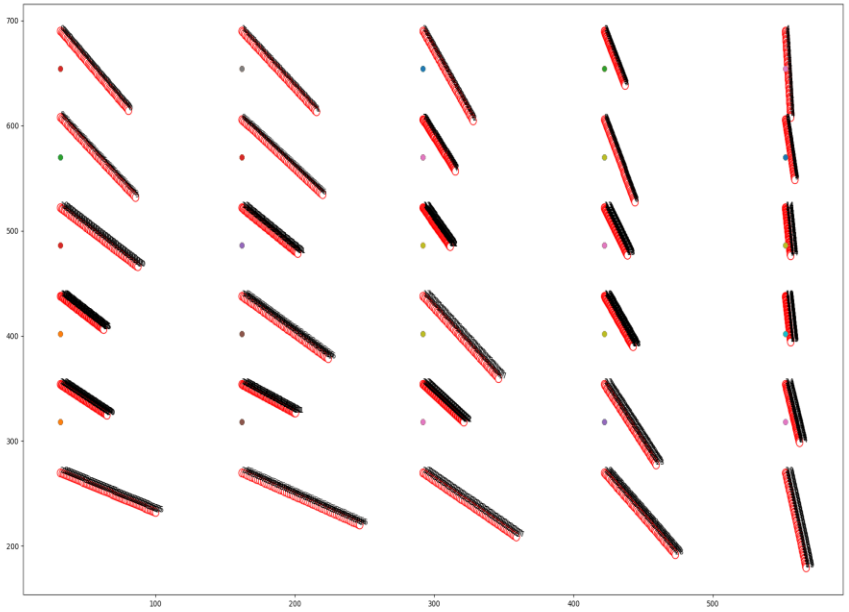



A t=4s

# Influence du coefficient de répulsion K:



$K=1.2 \times 10^5$



$K=1.2 \times 10^2$

# Discussion sur l'efficacité du modèle par comparaisons avec des résultats expérimentaux

Simulations d'exercice d'évacuation d'une salle carré de 5 m de coté, contenant 20 piétons de masse(min=60, max=100) et de pas de temps $h=10^{-2}$

| | Simulations | Expérience réelle |
|---|---|---|
| Temps d'évacuation | 7.9 s | 7.5 s |

La différence entre les résultats de l'exercice réel et ceux des simulations numériques est acceptable.

## ➢ Exemple 3:

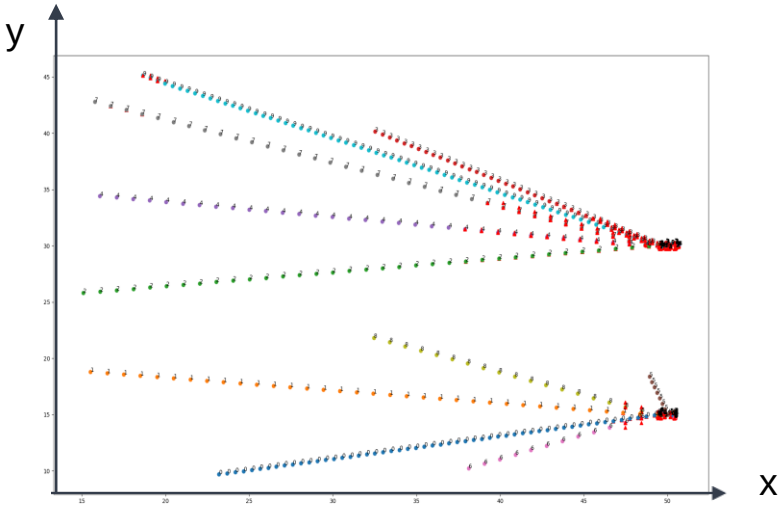



Figure 5: évacuation d'une salle avec 2 portes

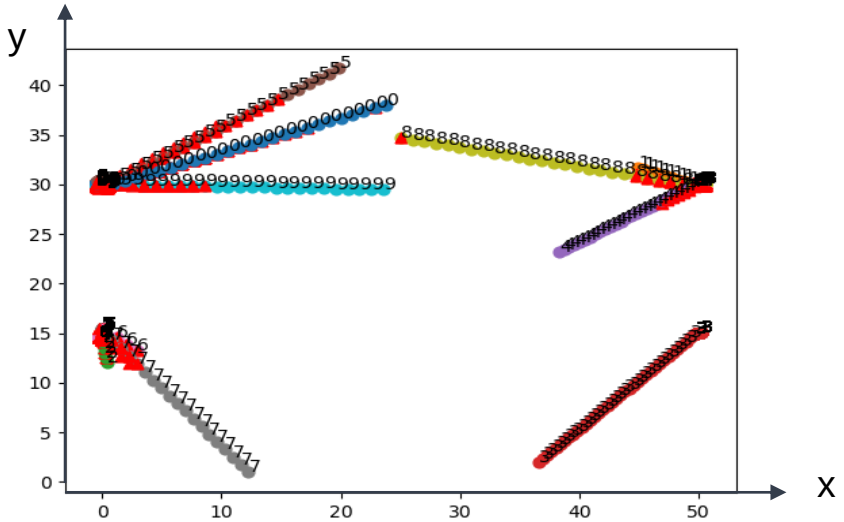
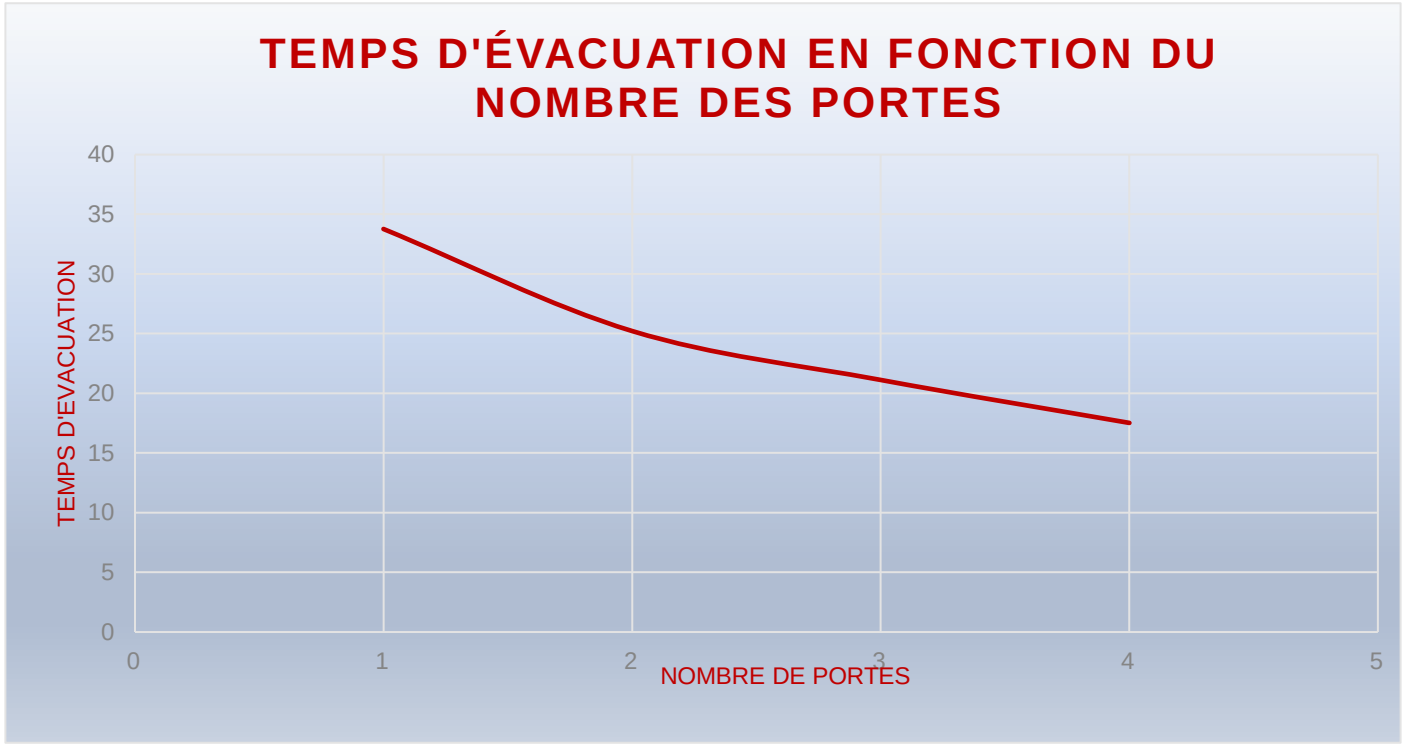



Figure 6: évacuation d'une salle avec 4 portes

| | 4 portes | 2 portes |
|---|---|---|
| Temps d'évacuation de 10 agents | 17.5 s | 25.2 s |

Figure 7: Comparaison de temps d'évacuation pour 2 et 4 portes.


TEMPS D'ÉVACUATION EN FONCTION DU NOMBRE DES PORTES
TEMPS D'EVACUATION
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
NOMBRE DE PORTES

## ➢ La sécurité des personnes:

Ce modèle sert pour:

- ✓ Comparer différentes solutions de conception des cheminements d'évacuation.
- ✓ mieux calibrer le nombre et la taille des voies d'évacuation.
- ✓ Identification des points de congestion …

Ce qui permet d':

- ✓ adapter le positionnement des agents de sécurité pendant l'organisation des évènements et de leur faire comprendre le risque identifié.
- ✓ offrir une qualité de sécurité optimale dans les futures et anciens constructions publiques et bâtiments.

# 4- Conclusions:

# ➢ ANNEXE:

- **Programme en absence d'obstacles:**

```
import matplotlib.pyplot as plt
import numpy as np
import math
import random

#%% Définition des données initiales
nombreAgents=20
VitesseInitiale=np.linspace(0,0,nombreAgents)
largeurPiece=50;
plt.xlim([0, largeurPiece]); plt.ylim([0, largeurPiece])
#génération aléatoire des coordonnées entre 0 et largeurPiece
coordonnees=np.zeros((nombreAgents,2))
for i in range(0, nombreAgents):
    coordonnees[i,0]=(random.uniform(0, 50))
    coordonnees[i,1]=(random.uniform(0, 50))
vitesseDesiree=np.random.randint(low=1, high=3, size=nombreAgents)
masse=np.random.randint(low=45, high=100, size=nombreAgents)
Tho=0.5 #temps de relaxation
listePortes=[[50,25]] #définition des coordonnées des portes
```

```
#%%Traçage conditions initiales
plt.scatter(listePortes[0][0],listePortes[0][1],color= 'black')
plt.show()

#%% Détection de la porte la plus proche
def getDistanceToDoor(numeroAgent,coordsPorte): #coordsPorte doit etre une liste de 2 [x,y]
        return (math.sqrt( (coordsPorte[0]-coordonnees[numeroAgent][0])**2 +(coordsPorte[1]-
       coordonnees[numeroAgent][1])**2))

def setClosetsDoor(numeroAgent,listePorte): #dans le cas de plusieurs portes il faut calculer la plus proche
    distanceToDoor = []
    for p in range(0,len(listePorte)):
            distanceToDoor.append(getDistanceToDoor(numeroAgent,listePorte[p]))
    return distanceToDoor.index(min(distanceToDoor))

#%% calcul direction désirée

def setDirection(numeroAgent):
    numeroPorteProche=setClosetsDoor(numeroAgent,listePortes)
    direction_x=listePortes[numeroPorteProche][0]- coordonnees[numeroAgent,0];
    direction_y=listePortes[numeroPorteProche][1] -coordonnees[numeroAgent,1]
    return
       np.array([direction_x/math.sqrt(direction_x**2+direction_y**2),direction_y/math.sqrt(direction_x**2+direction_y**2)])
```

```
plt.clf()
ax = plt.axes()
for i in range(0,nombreAgents):
    Dir=setDirection(i)
#%% Calcul de la force g et détection de contact
R=3 # pour améliorer le modèle on calcule la force g pour les plus proches voisins
def detectionContact (n,coordonneesTemps):
    result = []
    contactAgnts=0; agentsTouches=[]; XagentsTouches=[]; YagentsTouches=[]
    for m in range(0,n): #m pour les autres agents
        if m!= n and abs(coordonneesTemps[n,0]-coordonneesTemps[m,0])<R and
abs(coordonneesTemps[n,1]-coordonneesTemps[m,1])<R :
            agentsTouches.append(m)
            XagentsTouches.append(coordonneesTemps[m,0])
            YagentsTouches.append(coordonneesTemps[m,1])
            plt.scatter(coordonneesTemps[n,0],coordonneesTemps[n,1],marker='^',color='red')
            plt.scatter(coordonneesTemps[m,0],coordonneesTemps[m,1],marker='^',color='red')
            contactAgnts=1
    if agentsTouches !=[]:
        result =  [agentsTouches, XagentsTouches,  YagentsTouches]
```

```
else :
        result = []
    return result
def calculG(i,coordonneesTemps): #agent numéro i
    xx = detectionContact(i,coordonneesTemps)
    Gxi=0; Gyi=0
    if xx != []:
        k=10**2
        list_j=(xx)[0] #liste contenant les numéros des agents touchés
        list_Xj=(xx)[1]
        list_Yj=(xx)[2]
        Xi=coordonneesTemps[i,0]
        Yi=coordonneesTemps[i,1]
        r=0.5
        for j in range(0,len(list_Xj)):
            Dij=math.sqrt((Xi-list_Xj[j])**2+(Yi-list_Yj[j])**2)
            if Dij<=2*r:
                Gxi=Gxi+k*(Dij-2*r)*math.cos((Xi-list_Xj[j])/Dij)
                Gyi=Gyi+k*Dij*math.sin((Yi-list_Yj[j])/Dij)
            else:
                Gxi=Gxi-k*(Dij-2*r)*math.cos((Xi-list_Xj[j])/Dij)
                Gyi=Gyi-k*Dij*math.sin((Yi-list_Yj[j])/Dij)
    return (Gxi,Gyi)
 #%% Calcul de la force F
def calculF(numeroAgent,Vn):
    Dir=setDirection(numeroAgent)
    return masse[numeroAgent]*(vitesseDesiree[numeroAgent]* Dir-Vn)/Tho
```

```
#%% Calcul des vitesses et déplacements suivant x et y
h=0.01
#h est le pas du temps

def calculVitesse(numeroAgent,V_n,coordonneesTemps):
    return V_n+h/masse[numeroAgent] * (calculF(numeroAgent,V_n)+calculG(numeroAgent,coordonneesTemps) )   ;

def calculPosition (numeroAgent,V_n, coordonneesTemps):#position q_n et vitesse V_n sont des couples (,)
     V_nplus1= V_n+h/masse[numeroAgent] * (calculF(numeroAgent,V_n)+calculG(numeroAgent,coordonneesTemps))
    q_n=(coordonneesTemps[numeroAgent,0],coordonneesTemps[numeroAgent,1])
    return h*V_nplus1+ q_n

coordonneesNPls1=coordonnees; V_n=(0,0)
i=0
while min(coordonneesNPls1[:,0])<listePortes [0][0]:# pour assurer que tous les agents ont évacués la salle
    for N in range(0,nombreAgents):

        plt.text(coordonneesNPls1[N,0],coordonneesNPls1[N,1], str(N))
        plt.scatter(coordonneesNPls1[N,0],coordonneesNPls1[N,1])

        V_n=calculVitesse(N,V_n,coordonneesNPls1)
        coordonneesNPls1[N,0]=(calculPosition (N,V_n, coordonneesNPls1))[0]
        coordonneesNPls1[N,1]=(calculPosition (N,V_n, coordonneesNPls1))[1]
    i=i+1
    print(i) # pour determiner le temps d'evacuiation on multiplie i final par h le pas du mouvement.
```

- **Définition des obstacles pour l'exemple 2 et l'ajout de la force K:**

```
x1=[32,162,292,422,552]
Y1=[654,654,654,654,654]
pyplot.scatter(x1, Y1, s = 3000, c = 'darkblue', marker = 's')
plt.show()
x2=[32,162,292,422,552]
Y2=[570,570,570,570,570]
pyplot.scatter(x2, Y2, s = 3000, c = 'darkblue', marker = 's')
plt.show()
x3=[32,162,292,422,552]
Y3=[486,486,486,486,486]
pyplot.scatter(x3, Y3, s =3000, c = 'darkblue', marker = 's')
plt.show()
x4=[32,162,292,422,552]
Y4=[402,402,402,402,402]
pyplot.scatter(x4, Y4, s = 3000, c = 'darkblue', marker = 's')
plt.show()
x5=[32,162,292,422,552]
Y5=[318,318,318,318,318]
pyplot.scatter(x5, Y5, s =3000, c = 'darkblue', marker = 's')
plt.show()
x6=[32,162,292,422,552]
Y6=[234,234,234,234,234]
pyplot.scatter(x6, Y6, s = 3000, c = 'darkblue', marker = 's')
plt.show()
```

```
def detectionContactObstacles (n,coordonneesTemps):
#L liste des positions des obstacles prédéfinies précédemment
    result = []
    contactAgnts=0; agentsTouches=[]; XagentsTouches=[]; YagentsTouches=[]
    for m in range(0,n): #m pour les obstacles
        if abs(coordonneesTemps[n,0]-L[m][0])<l and abs(coordonneesTemps[n,1]-L[m][1])<R :
            agentsTouches.append(m)
            XagentsTouches.append(L[m][0])
            YagentsTouches.append(L[m][1])

            plt.scatter(L[m][0],L[m][1])
            contactAgnts=1

    if agentsTouches !=[]:
        result =  [agentsTouches, XagentsTouches,  YagentsTouches]
        YagentsTouches,[coordonneesTemps[n,0],coordonneesTemps[n,1]]]
    else :
        result = []
    return result
```

```
def calculK(i,coordonneesTemps): # force de contact entre agent numéro i et les obstacles.
    xx = detectionContactObstacles(i,coordonneesTemps)
    Kxi=0; Kyi=0
    if xx != []:
        k=1.2*10**5
        list_j=(xx)[0] #liste contenant les numéros des agents touché
        list_Xj=(xx)[1]
        list_Yj=(xx)[2]
        Xi=coordonneesTemps[i,0]
        Yi=coordonneesTemps[i,1]
        r=0.5
        for j in range(0,len(list_Xj)):
            Dij=math.sqrt((Xi-list_Xj[j])**2+(Yi-list_Yj[j])**2)
if Dij<=r+20:
                Kxi=Kxi+k*(Dij-r-20)*math.cos((Xi-list_Xj[j])/Dij)
                Kyi=Kyi+k*(Dij-r-20)*math.sin((Yi-list_Yj[j])/Dij)
            else:
                Kxi=Kxi-k*(Dij-r-20)*math.cos((Xi-list_Xj[j])/Dij)
                Kyi=Kyi-k*(Dij-r-20)*math.sin((Yi-list_Yj[j])/Dij)
    return (Kxi,Kyi)
```