TIPE 2022-2023

MODÉLISATION DES MOUVEMENTS DE FOULE

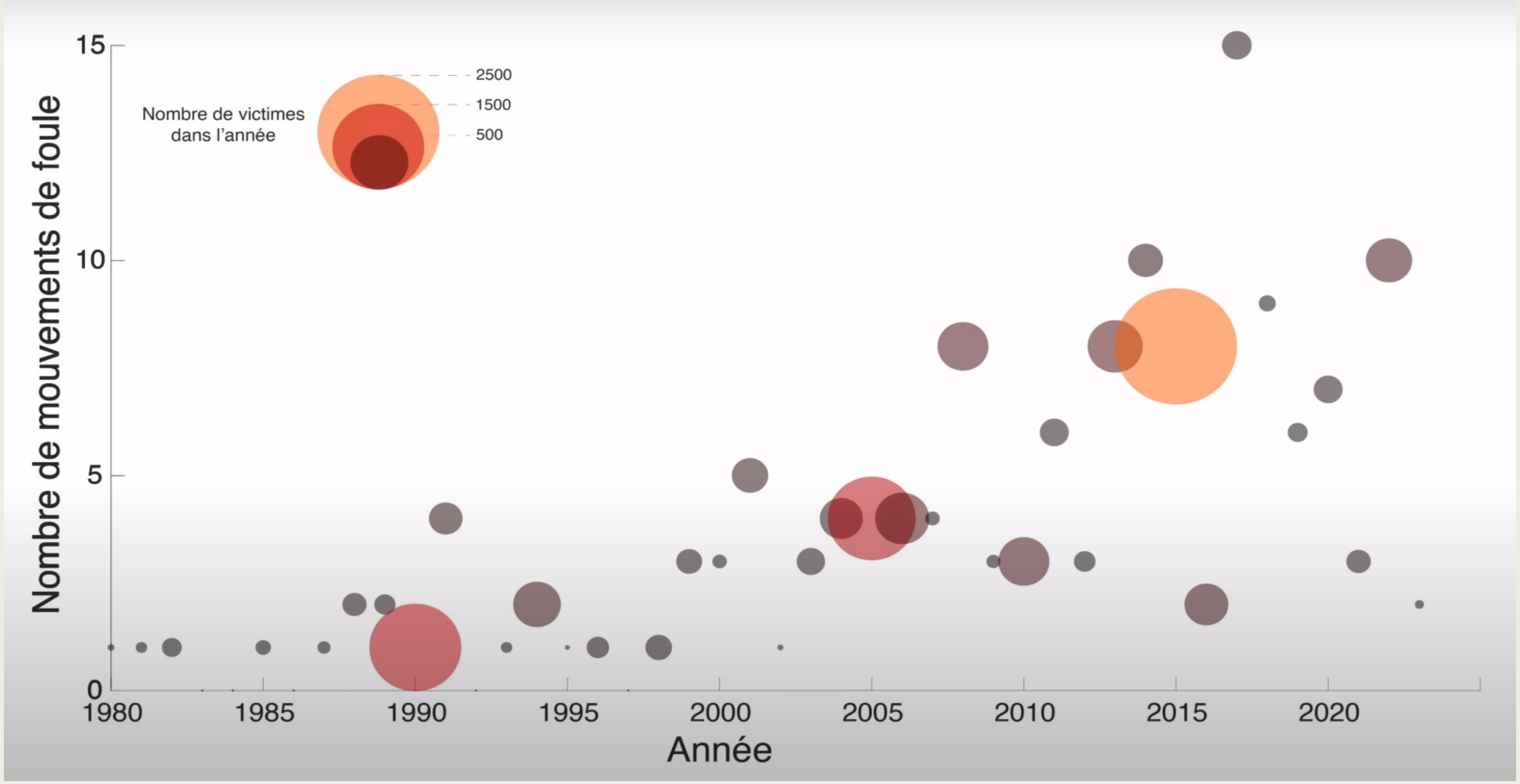
Thème: La ville

MALLEZ Alexandre - 41866

Sommaire:

Introduction

1



Statistiques répertoriées et publiées en 2023 par Mr Moussaid, chercheur à à l'université Max Planck spécialiste des foules

1

Problématisation

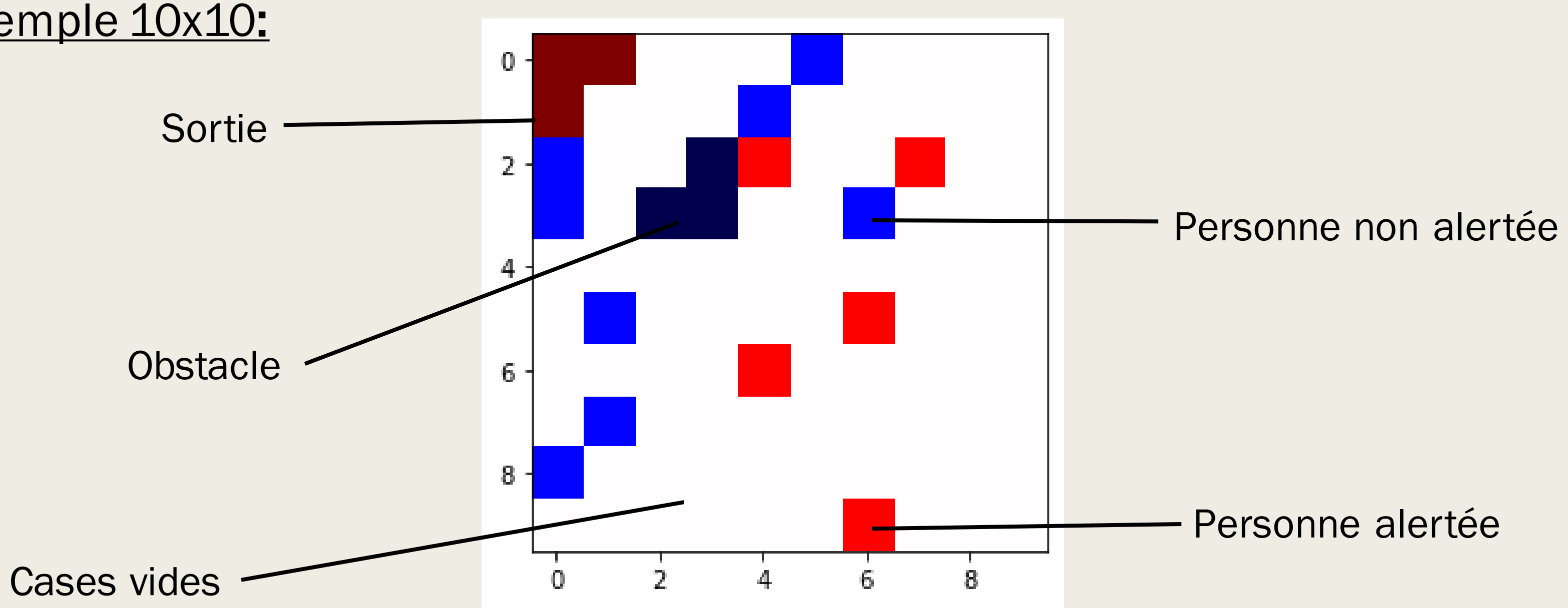
- **Cadre:** évacuation d'une foule

- **But :** Optimisation de la vitesse d'évacuation

- **Méthode :** Mise en œuvre du *"Slower is Faster effect"* à travers différents modèles

Automates cellulaires :

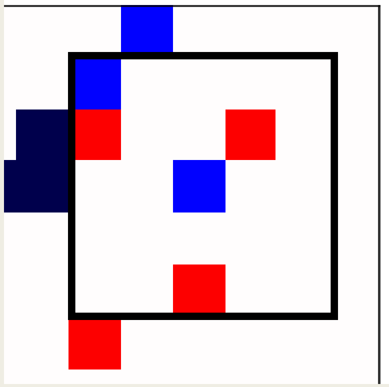
Exemple 10x10:



Notion de voisinage:

-Probabilité de devenir alerté

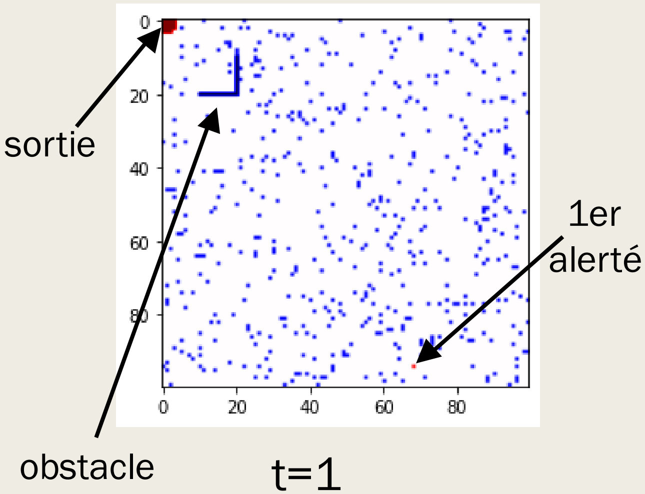
-Inertie



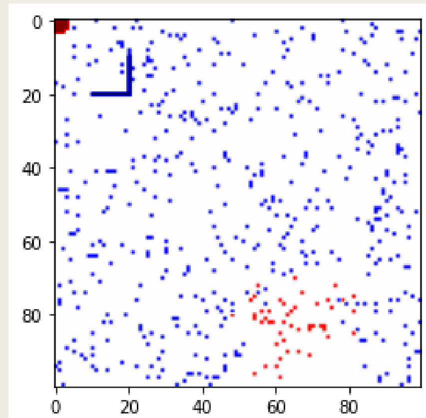
2

Simulation :

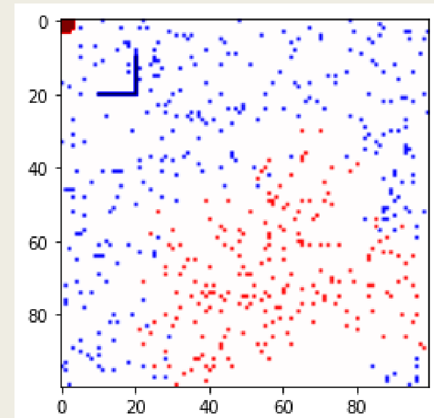
Taille carte: 100x100 Population: 500 Voisinage: 8



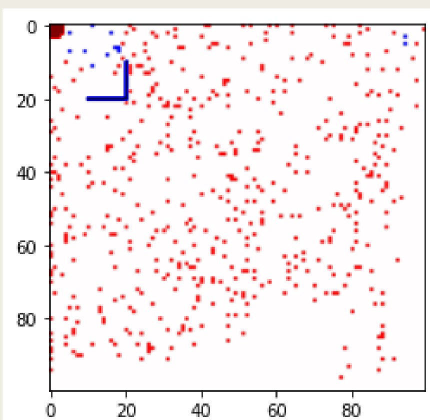
t=1



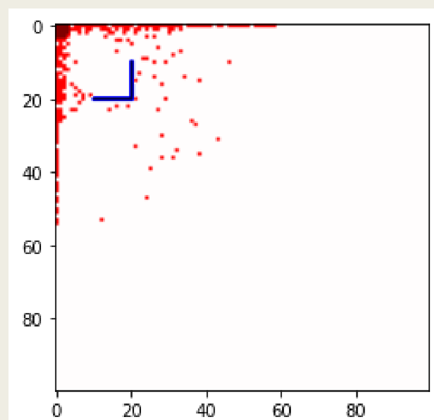
t=10



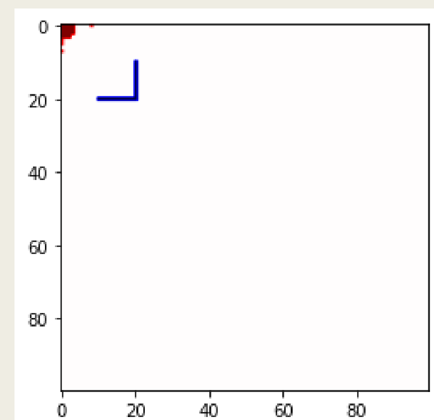
t=20



t=40



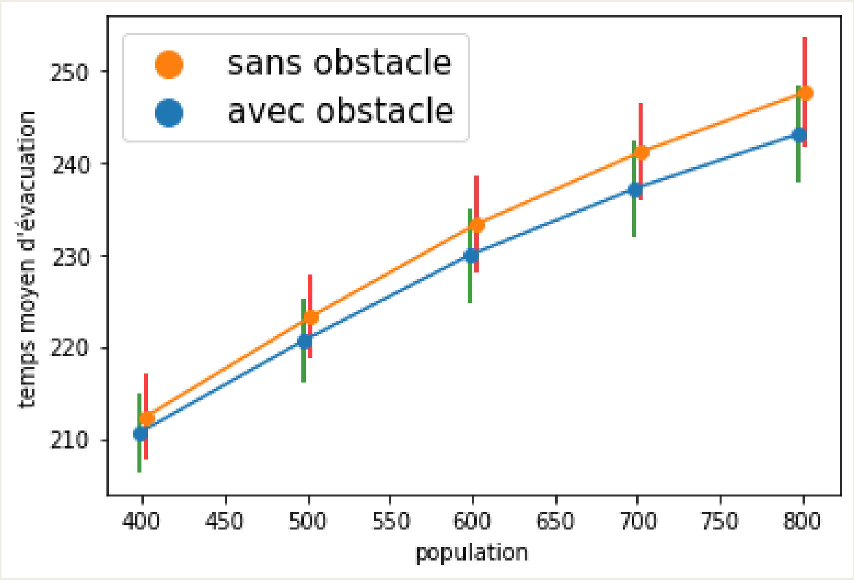
t=120



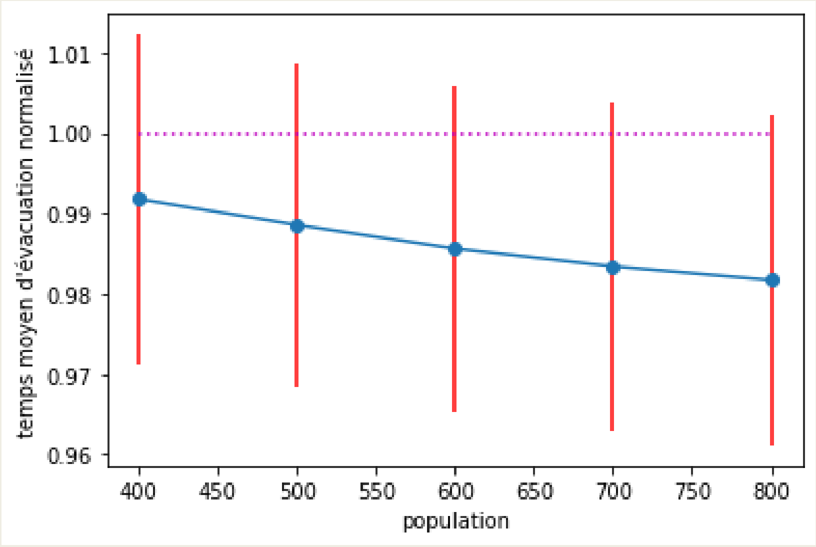
t=205

Résultats :

2



Taille carte: 100x100
Voisinage: 8
Population: variable



Baisse du temps d'évacuation allant jusqu'à 2%

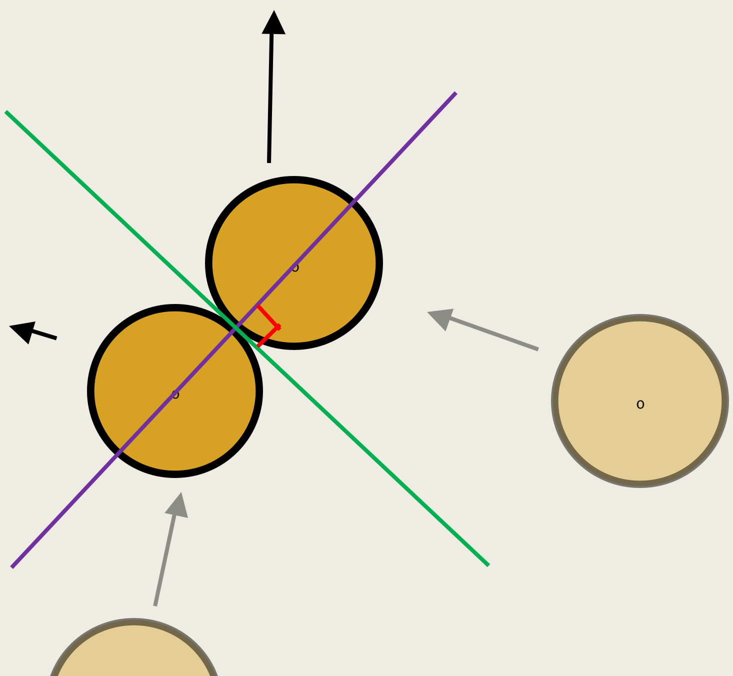
Incertitude élevé

3

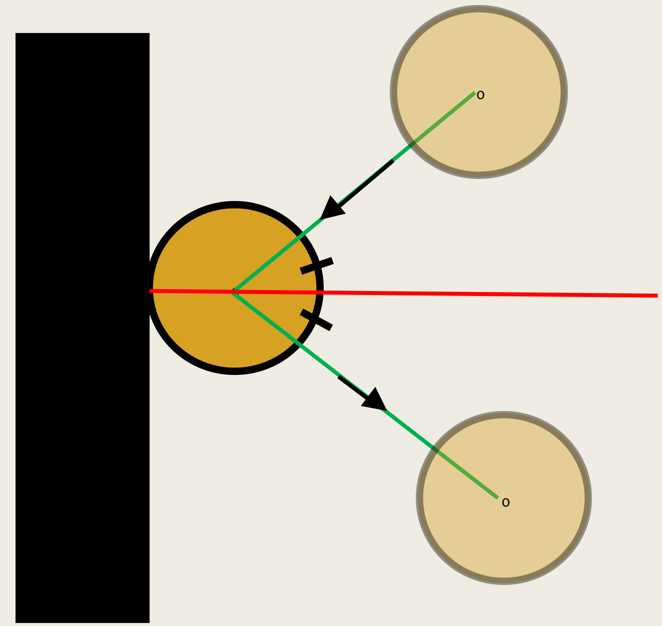
Modèle granulaire:

Principe : assimilation de la population à des boules/grains soumis au poids seul

Contact boule-boule



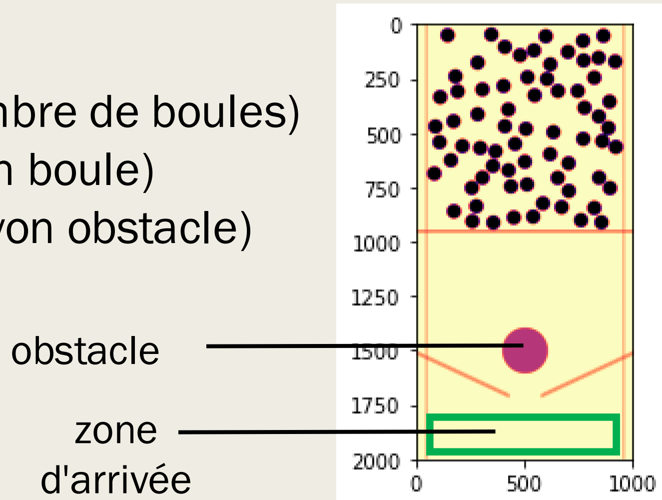
Contact mur/boule et obstacle/boule



3

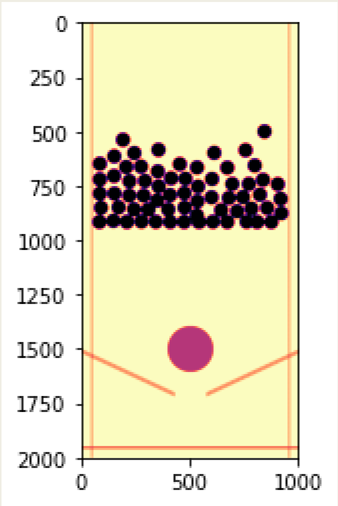
Simulation informatique:

g=10
N=70 (nombre de boules)
r=33 (rayon boule)
R=105 (rayon obstacle)

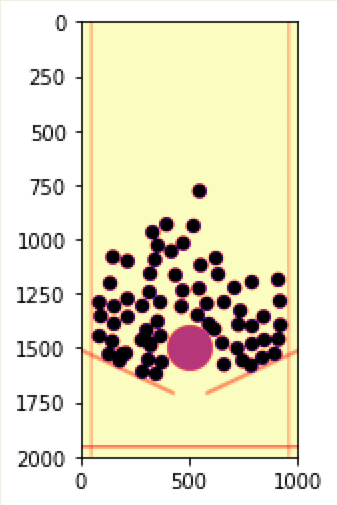


t=0

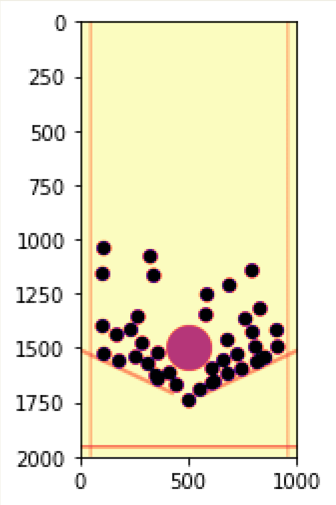
1ère étape de
stabilisation
d'un tas :



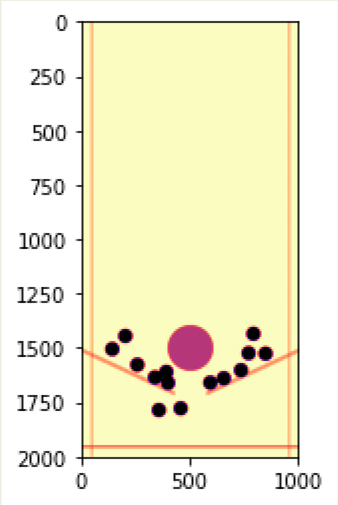
t=10000



t=16000



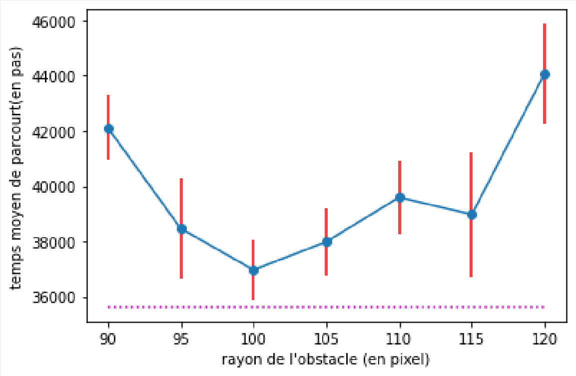
t=30000



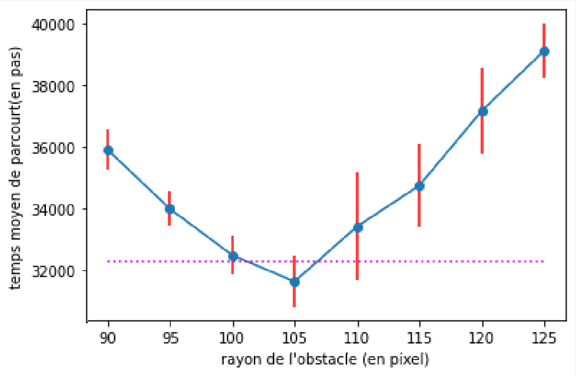
t=42000

Résultats:

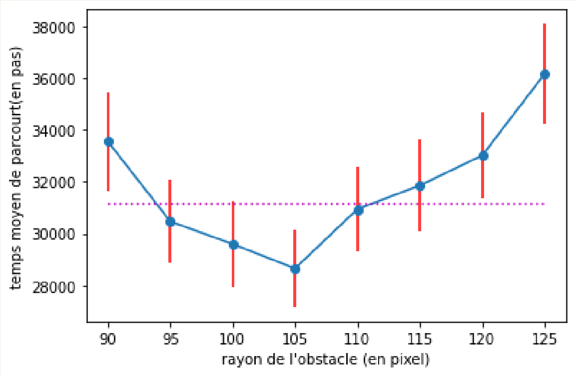
3



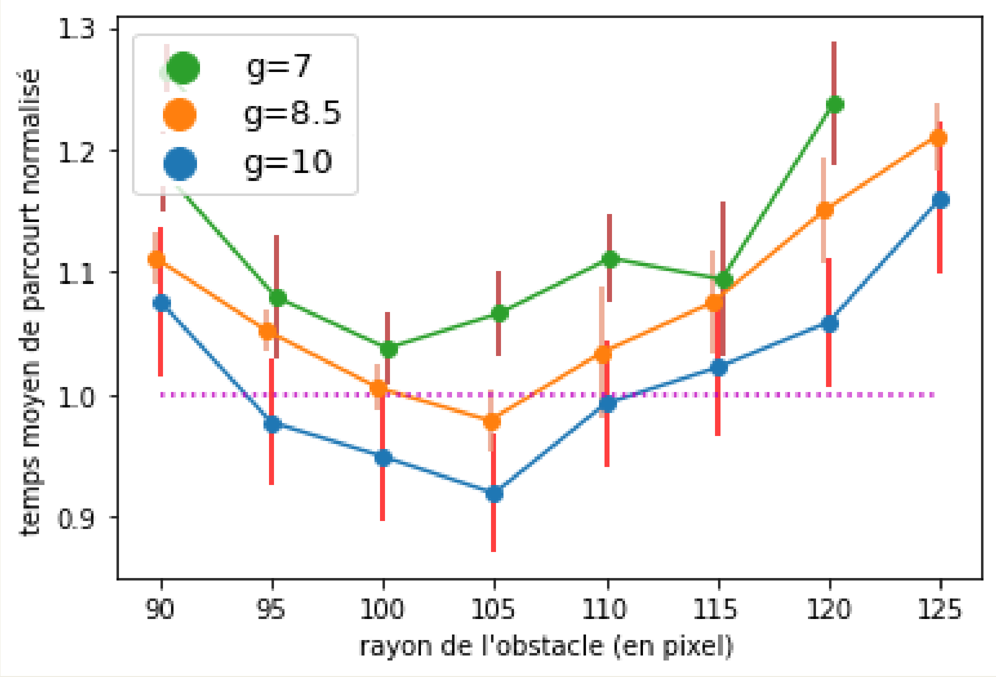
g=7



g=8.5



g=10

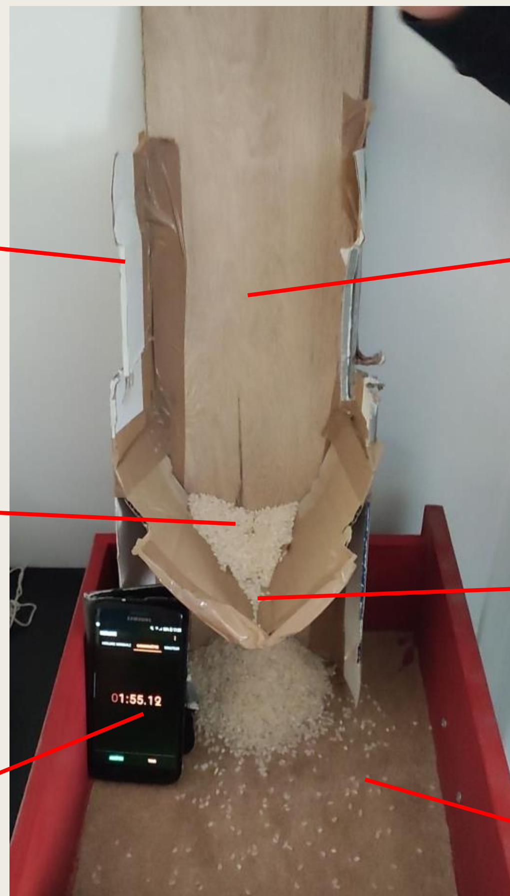


Le rayon optimal de l'obstacle est 3,2x celui d'une boule (r=105)

4

Simulation expérimentale:

Expériences réalisées avec
100g de riz cru



Bordures latérales

rampe de glissement

riz

ouverture

chronomètre

bac de réception

Angle θ variable

Simulation expérimentale:

Expérience avec obstacle et un angle θ=22°:

1

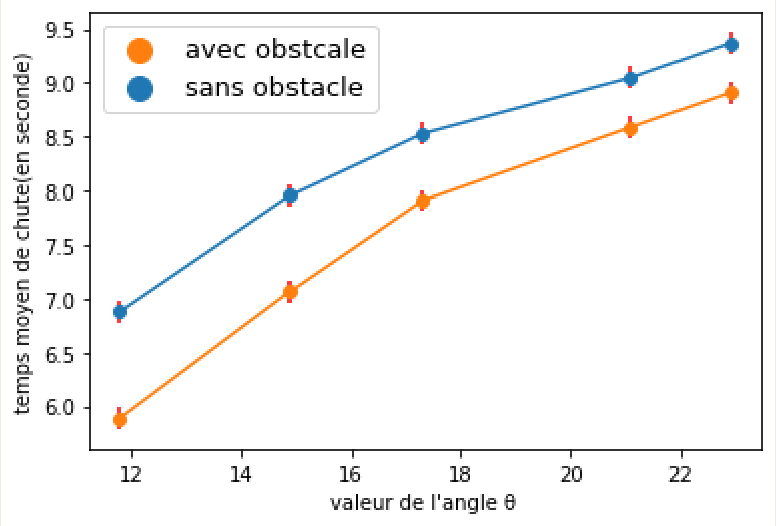
2

3

4

Résultats:

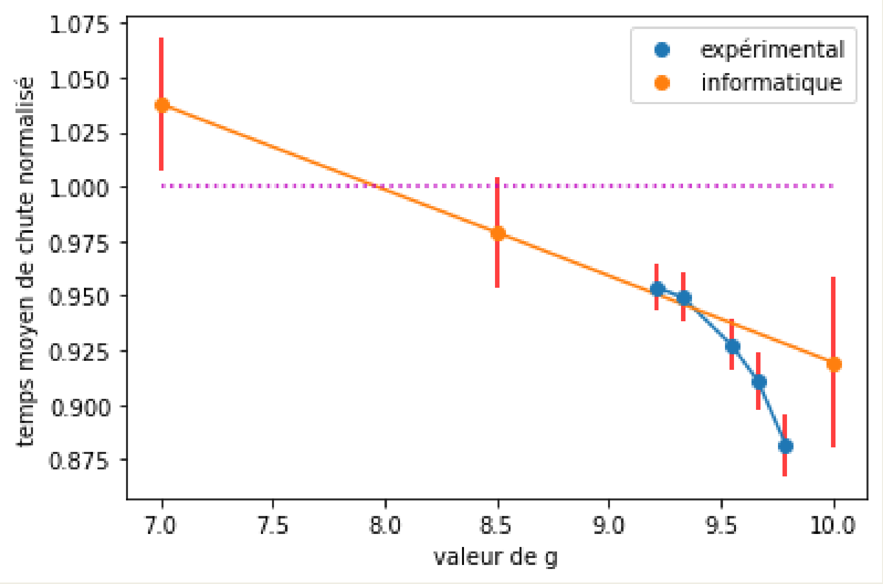
4



m=100g de riz
θ variant
Dimension des pentes, ouvertures et obstacle constante.

Comparaison avec le modèle informatique avec le rayon d'obstacle optimal:

$g=9.8\cos\theta$



5

Conclusion:

Solution retenue:

Obstacle circulaire (du moins arrondie) situé devant la sortie

Efficace à forte densité et accélération : réduction jusqu'à 10%

Pistes d'améliorations:

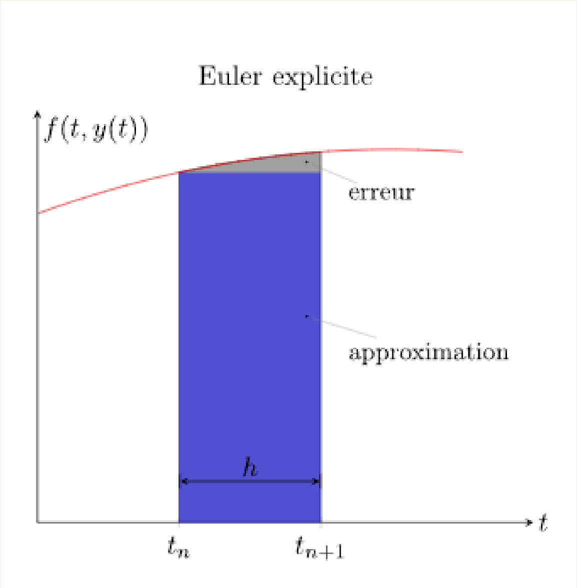
-Essayer plus d'obstacles différents

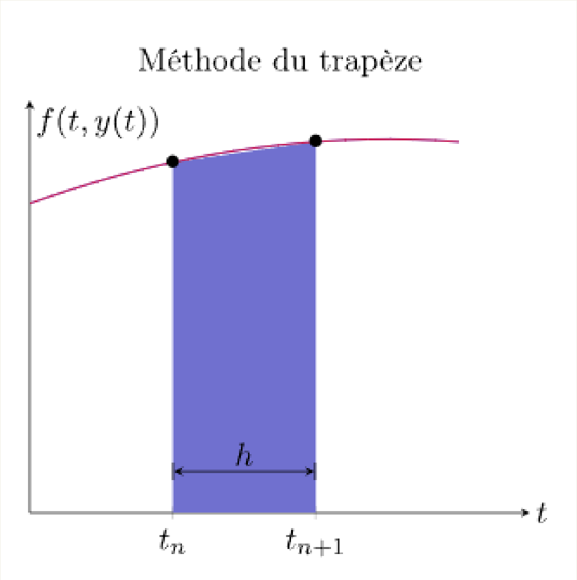
-Expérience à échelle humaine pour comparer les résultats

-Préciser les modèles en rajoutant des forces/effets réelles observées dans la réalité…

ANNEXE:

Méthode Runge Kutta 2 par rapport à Euler explicite :





v[n+1] = v[n] + a[n]*dt

a'[n]=(a[n+1]+a[n])/2
v[n+1] = v[n]+ a'[n]*dt

Forces sociales:

5

Représentation phénoménologique des forces appliquées sur chaque individu, représenté par une boule:

1) Force attirant l'individu vers la sortie	$\overrightarrow{F_i^S}(t) = \dfrac{\overrightarrow{v_{s,i}}(t) - \overrightarrow{v_i}}{\tau_i}$	$\overrightarrow{v_{s,i}}(t)$ la vitesse souhaitée
2) Force repoussant l'individu appliqué par les murs	$\overrightarrow{F_{k\rightarrow i}^M}(t) = -\overrightarrow{grad}(U_k)(\overrightarrow{r_i}(t))$ $U_k(\overrightarrow{r}) = U^0 \exp(-\frac{d(\overrightarrow{r}, M_k)}{\mu})$	
3) Force d'interaction entre les individus	$\overrightarrow{F_{j\rightarrow i}^P}(t) = -\overrightarrow{grad}(V_j)(\overrightarrow{r_i}(t))$	

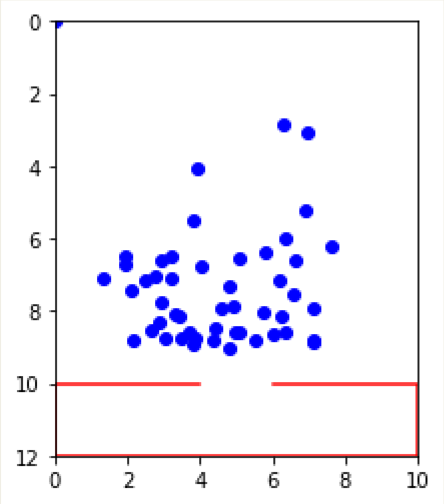
$$V_j(\overrightarrow{r}) = V^0 \exp(-\frac{b_j(\overrightarrow{r})}{\sigma}) \quad b_j(\overrightarrow{r}) = \frac{\sqrt{(\|\overrightarrow{r} - \overrightarrow{r_j}\| + \|\overrightarrow{r} - \overrightarrow{r_j} - v_j t.\overrightarrow{e_j}\|)^2 - (v_j t)^2}}{2}$$

Simulation :

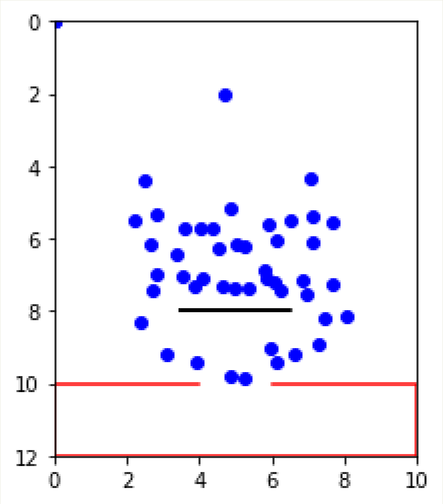
5

Population: 50

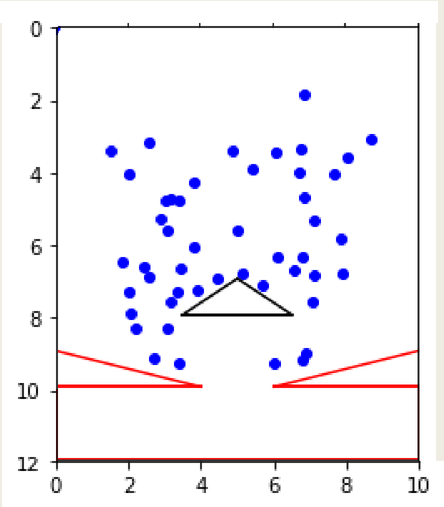
3 types :



Sans obstacle



Obstacle plat



Obstacle plat
et pente

Résultats:

5

Forme	Moyenne	Ecart-type
Sans obstacle	20818	710
Avec obstacle plat	22144	921
Avec obstacle triangulaire	19237	653

Code automates cellulaires:

```python
import numpy as np
import matplotlib.pyplot as plt
import random as r

cpt=0
while cpt<1:
    n=10 #taille map
    h=10 #nombre de points/personnes sur la map
    v=10#perimetre de propagation
    s=2 #taille sortie
    N=500 #nombre d'étape

    M=[[0 for _ in range(n)] for _ in range(n)]
    d={}
    for j in range(2,4):
        M[3][j]=4
        M[j][3]=4
    for i in range(s):
        for j in range(s):
            M[i][j]=3
    M[s-1][s-1]=0

    for i in range(h+1):
        x,y=r.randint(0,n-1),r.randint(0,n-1)
        while M[x][y]!=0:
            x,y=r.randint(0,n-1),r.randint(0,n-1)
        d[i]=(x,y,0,0,0)  #position,état,voisins,voisins alértés
        M[x][y]=1


    def compteur_voisin(d,M):
        for i in d:
           x,y,a,b,c=d[i]
           cpt=0
           alert=0
           for x0 in range (2*v+1):
               if x-v+x0>=0 and x-v+x0<n:
                   for y0 in range (2*v+1):
                       if y-v+y0>=0 and y-v+y0<n:
                           if (x-v+x0,y-v+y0)!=(x,y):
                              if M[x-v+x0][y-v+y0]!=0 and M[x-v+x0][y-v+y0]!=3 and M[x-v+x0][y-v+y0]!=4:
                                 cpt+=1
                              if M[x-v+x0][y-v+y0]==2:
                                  alert+=1
           d[i]=(x,y,a,cpt,alert)
        return d

    def maj_etat(d,M):
        for i in d:
           x,y,a,b,c=d[i] #b,c=voisins,voisins alertés
           if a!=3:
             if b!=0:
                P=[0 for _ in range (b)]
                for j in range (c):
                   P[j]=1
                if P[r.randint(0,b-1)]==1:
                   M[x][y]=2
                   d[i]=(x,y,1,b,c)
        return d,M

    def deb(d,i):
       A,B,C,D,E=d[0]
       M[A][B]=2
       d[i]=(A,B,1,D,E)
       return d
```

5

```python
    def random_move(d,M):
        for k in d:
         x,y,a,b,c=d[k]
         if a==1:
            m=r.randint(0,1)
            i,j=x,y
            if x==0 and M[x][y-1]==0:
                m=2
                j-=1
            if y==0 and M[x-1][y]==0:
                m=2
                i-=1
            if x!=0 and m==1:
               if M[x-1][y]==0:
                   i-=1
               elif y!=0 and M[x][y-1]==0:
                   j-=1
            if y!=0 and m==0:
                if M[x][y-1]==0:
                  j-=1
                elif x!=0 and M[x-1][y]==0:
                   i-=1
            if M[x][y-1]==3 or M[x-1][y]==3 :
                d[k]=i,j,3,b,c
                M[x][y]=0
            if (i,j)!=(x,y):
                d[k]=(i,j,a,b,c)
                M[i][j]=M[x][y]
                M[x][y]=0
        return d,M

    def last(M,i):
        cpt=0
        for k in range(n):
            for j in range(n):
                if M[k][j]==1 or M[k][j]==2:
                    cpt+=1
        if cpt==0:
            return i
        return N

    def couleur(M):
        M1=[[0 for _ in range(len(M[0]))]for _ in range(len(M))]
        for i in range(len(M)):
            for j in range(len(M[i])):
                if M[i][j]==0:
                    M1[i][j]=2
                if M[i][j]==1:
                    M1[i][j]=1
                if M[i][j]==2:
                    M1[i][j]=3
                if M[i][j]==3:
                    M1[i][j]=4
                if M[i][j]==4:
                    M1[i][j]=0
        return M1

    def show(n, M,d):
        plt.figure()
        d=deb(d,0)
        k=N
        i=0
        while i<=N and k==N:
            M1=couleur(M)
            plt.imshow(M1,cmap='seismic')
            i+=1
            k=last(M,i)
            plt.show()
            d,M=maj_etat(d,M)
            d,M=random_move(d,M)
            d=compteur_voisin(d,M)
        print(k)
    cpt+=1
    show(n,M,d)
```

Code modèle granulaire 1:

5

```
import matplotlib.pyplot as plt
import random as r

g=10 #g
N=1000 #taille map
n=70 #nombre de boules
R=N//30 #rayon boules
L=N//20
M=[[0 for _ in range(N+1)]for _ in range(N+1)]
d={}
dt=0.002
t=100000 #nombre d'itération maximal
s=1 #decris le changement d'étape
hh=int(N*15/10) #hauteur barriere
p=2.15 #coeff de la pente
yo,xo=int(N/2),int(N*15/10) #position de l'obstacle
rr=105 #rayon obstacle

e=0.7 #coefficient d'amortissement boule/sol
e1=1 #coefficient d'amortissement boule/boule
def compatible(x,y,d):
    if (x,y)==(0,0):
        return False
    for j in d:
        a,b,_,_=d[j]
        if ((x-a)**2 +(y-b)**2)< 4* R**2:
            return False
    return True

for l in range(n):
    x,y=0,0
    while compatible(x,y,d)==False:
        x,y=r.randint(R,N-R-L),r.randint(L+R,N-R-L)
    d[l]=(x,y,(0,0),(0,0))

########
def poids(d):
    for j in d:
        x,y,v,a=d[j]
        ax,ay=a
        ax+=g
        a=ax,ay
        d[j]=x,y,v,a
    return d


def maj_vitesse(d,d1):
    for j in d:
        x1,y1,v1,a1=d1[j]
        x,y,v,a=d[j]
        vx1,vy1=v1
        vx,vy=v
        ax1,ay1=a1
        ax,ay=a
        v=(vx+(ax+ax1)*dt/2,vy+(ay+ay1)*dt/2)
        d[j]=x,y,v,(ax,ay)
    return d
```

```
def maj_position(d,d1):
    for j in d:
        x1,y1,v1,a1=d1[j]
        x,y,v,a=d[j]
        vx1,vy1=v1
        vx,vy=v
        x=x+(vx+vx1)*dt/2
        y=y+(vy+vy1)*dt/2
        d[j]=x,y,v,a
    return d

def bords_bas(d,s):
    for j in d:
        x,y,v,_=d[j]
        vx,vy=v
        if x>s*N-R-L and vx>2:
                vx=-e*vx
                v=(vx,vy)
                d[j]=x,y,v,_
        elif x>s*N-R-L and vx>-1 and vx<2:
                d[j]=x,y,(0,vy),(0,0)
        else:
                d[j]=x,y,v,(g,0)
        if y>N-R-L and vy>0.5:
            v=(vx,e*-vy)
            d[j]=x,y,v,_
        if y<R+L and vy<0.5:
            v=(vx,e*-vy)
            d[j]=x,y,v,_
        if (y>N-R-L or y<R+L) and vy>-0.5 and vy<0.5:
            d[j]=x,y,(vx,0),_
        if s==2:
           if abs(x - (hh-20) - y/p) <=2 and y<N*p/5 and y>10:
              q=np.arctan(p)
              va=vx*np.cos(q)+vy*np.sin(q)
              vb=-vx*np.sin(q)+vy*np.cos(q)
              vb=-e*vb
              vx=va*np.cos(q)-vb*np.sin(q)
              vy=va*np.sin(q)+vb*np.cos(q)
              v=(vx,vy)
              d[j]=x,y,v,_
           if abs(x - (hh-20) - (N-y)/p) <=2 and y> N - N*p/5 and y<N-10:
                q=np.arctan(-p)
                va=vx*np.cos(q)+vy*np.sin(q)
                vb=-vx*np.sin(q)+vy*np.cos(q)
                vb=-e*vb
                vx=va*np.cos(q)-vb*np.sin(q)
                vy=va*np.sin(q)+vb*np.cos(q)
                v=(vx,vy)
                d[j]=x,y,v,_
    return(d)
```

Code modèle granulaire 2 :

5

```python
def obstacle(d):
    for j in d:
        x,y,v,a=d[j]
        x1,y1,v1=xo,yo,(0,0)
        if ((x-x1)**2 +(y-y1)**2)<(R+rr)**2:
            if y!=y1:
                q=np.arctan((x1-x)/(y1-y))
            else:
                q=np.pi
            vx,vy=v
            vx1,vy1=v1
            va=-vx*np.cos(q) + vy*np.sin(q)
            vb=vx*np.sin(q) + vy*np.cos(q)
            vb=-vb
            vx=-va*np.cos(q) + vb*np.sin(q)
            vy=va*np.sin(q) + vb*np.cos(q)
            v=(vx,vy)
            d[j]=x,y,v,a
    return d

def maj_matrice(d,s):
    M=[[0 for _ in range(N+1)]for _ in range(2*N+1)]
    for i in range(0,N):
        for t in range(8):
          M[s*N-L+t][i]=1
          M[2*i][N-L+t]=1
          M[2*i][L-t]=1
    for i in range (hh,hh+(int(2*N/10) )):
        j=int(i*p - hh*p)
        for t in range(20):
          M[i+t][j]=1
          M[i+t][-j]=1
    for j in d:
        x,y,_,_=d[j]
        x,y=int(x),int(y)
        for i in range(x-R,x+R):
          for j in range(y-R,y+R):
             if ((x-i)**2 +(y-j)**2)<R**2:
                 M[i][j]=2
    for i in range(xo-rr,xo+rr):
        for j in range(yo-rr,yo+rr):
            if ((xo-i)**2 + (yo-j)**2)<rr**2:
              M[i][j]=1
    return M
```

```python
double_contact(d,s):
for j in d:
    c=-1
    for k in d:
        x,y,v,a=d[j]
        x1,y1,v1,a1=d[k]
        if ((x-x1)**2 +(y-y1)**2)<4*R**2:
            vx1,vy1=v1
            if x>s*N-R-L-10:
               if vx1**2+vy1**2<20:
                  c+=1
    if c==2:
        d[j]=x,y,(0,0),a
return d

contact(d,h,s):
f=[]
for j in d:
 if j not in f:
    for k in d:
       if j!=k and h[j,k]==0:
          x,y,v,a=d[j]
          x1,y1,v1,a1=d[k]
          if ((x-x1)**2 +(y-y1)**2)<4*R**2:
              f.append(k)
              h[j,k]=0
              if y!=y1:
                 q=np.arctan((x1-x)/(y1-y))
              else:
                  q=np.arccos(0)
              vx,vy=v
              vx1,vy1=v1
              va=-vx*np.cos(q) + vy*np.sin(q)
              va1= -vx1*np.cos(q) + vy1*np.sin(q)
              vb=vx*np.sin(q) + vy*np.cos(q)
              vb1=vx1*np.sin(q ) +vy1*np.cos(q)
              vb,vb1=vb1,vb
              va,va1=e1*va,e1*va1
              vx=-va*np.cos(q) + vb*np.sin(q)
              vy=va*np.sin(q) + vb*np.cos(q)
              vx1=-va1*np.cos(q) + vb1*np.sin(q)
              vy1=va1*np.sin(q) + vb1*np.cos(q)
              if x>s*N-L-R and vx>0:
                    vx=0
              if x1>s*N-L-R and vx1>0:
                    vx1=0
              v=(vx,vy)
              v1=(vx1,vy1)
              d[j]=x,y,v,a
              d[k]=x1,y1,v1,a1
return d,h
```

```python
def controle(d,s):
    Sup=[]
    for j in d:
        x,y,v,_=d[j]
        if x>9*N/5:
            Sup.append(j)
    for j in Sup:
        del(d[j]) #élimine les boules du bas
    return(d)
"""
Projections: si x1,y1>x,y
va1=vx1*np.cos(q)+vy1*np.sin(q)
vb1=-vx1*np.sin(q)+vy1*np.cos(q)
vx1=va1*np.cos(q)-vb1*np.sin(q)
vy1=va1*np.sin(q)+vb1*np.cos(q)
   """
h={}
for j in d:
    for k in d:
        h[(j,k)]=0

def maj_h(h):
    for i in h:
        if h[i]!=0:
            h[i]-=1
    return h

for l in range(n):
     x,y=0,0
     while compatible(x,y,d)==False:
         x,y=r.randint(R,N-R-L),r.randint(L+R,N-R-L)
     d[l]=(x,y,(0,0),(0,0))
d=poids(d)
d1=d.copy()
for i in range(t):
     h= maj_h(h)
     d1=d.copy()
     d=maj_position(d,d1)
     d,h=contact(d,h,s)
     d=maj_vitesse(d,d1)
     d=bords_bas(d,s)
     d=double_contact(d,s)
     d=controle(d,s)
     d=obstacle(d)
     if len(d)<int(n/10):
        print(i)
        break
     if i%500==0:
       M=maj_matrice(d,s)
       plt.figure()
       plt.imshow(M, cmap='magma_r')
       plt.show()
     if s==1 and i%10000==9999:
        s=2
        if s==2:
            d=poids(d)
```