Aulagnier Mathis

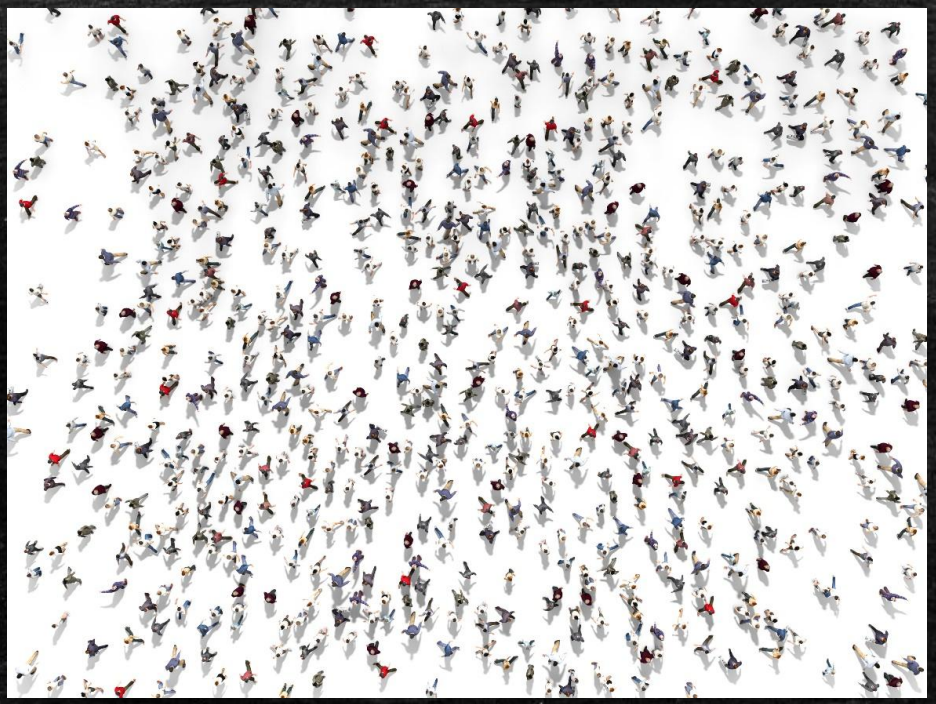
n° 33382

Compréhension et prédiction des mouvements de foules

TIPE – Santé, Prévention -

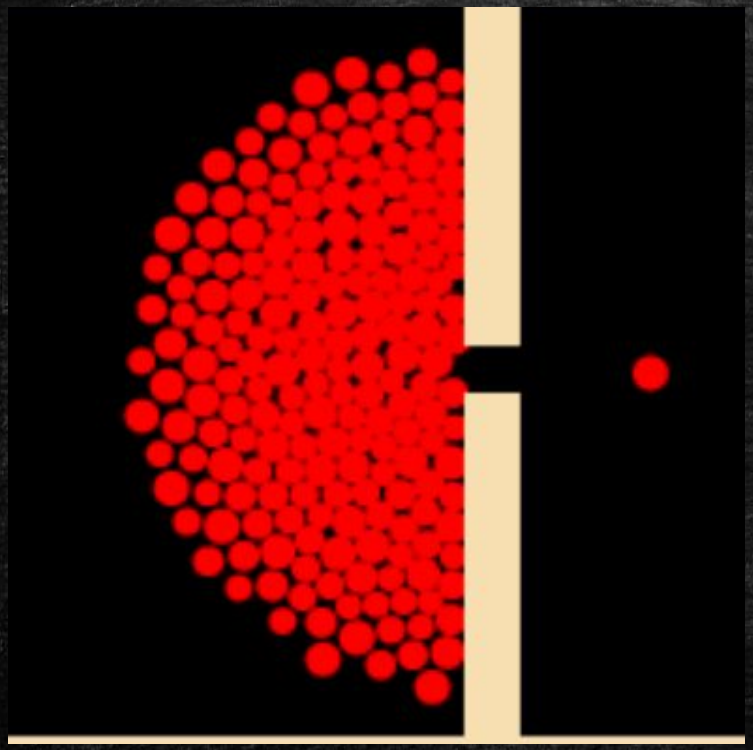
Problématique

Prévoir et sécuriser le mouvement d’une foule d’individus dans un lieu clos.



Source: https://www.pngegg.com/fr/png-ddwrl

Différents modèles de modélisation d'une foule



Source: Modelisation macroscopique de mouvements de foule, Aude Roudnef

Structure de la présentation

A. Conception d'un modèle microscopique
   a) Construction du modèle d'études
   b) Analyse des simulations obtenues

B. Fabrication d'une maquette
   a) Modèle imparfait
   b) Mise en lumière des effets d'arche

C. Amélioration de notre modèle en 2D
   a) Nouveautés apportées
   b) Optimisation des facteurs d'encombrement au niveau des sorties

Conception d’un modèle microscopique pour l'étude d’une foule

Phase initiale(1)

`creation_foule` prend en entrée un entier N et un réel `densite` et retourne un tableau `foule` de type `np.array` représentant une foule d'individus présente dans une enceinte carrée de capacité $(2N+1)^2$ avec une densité donnée par le réel `densite`.

Chaque élément `individu` de `foule` est un tableau de trois réels.

- La première valeur représente l'abscisse de la position de `individu`.
- La deuxième valeur représente l'ordonnée de la position de `individu`.
- La troisième valeur représente le statut de `individu` qui vaut
  - 0 tant que la personne est encore **présente** dans l'enceinte
  - 1 lorsque la personne est sortie de l'enceinte.

Exemple

```
>>> ff = creation_foule(10,0.2)
>>> ff[12]
array([-4., -5., 0.])
```

Le treizième individu, celui dont l'indice est 12 a pour abscisse -4, pour ordonnée -5 et pour statut 0.

Phase initiale(2)

On décrit aussi les fonctions suivantes qui nous seront utiles :

La fonction `creation_sorties` prend en entrée un entier N et retourne une liste de sorties présentes sur le contour du domaine.

La fonction `creation_evenements` prend en entrée un entier N et génère un évènement.

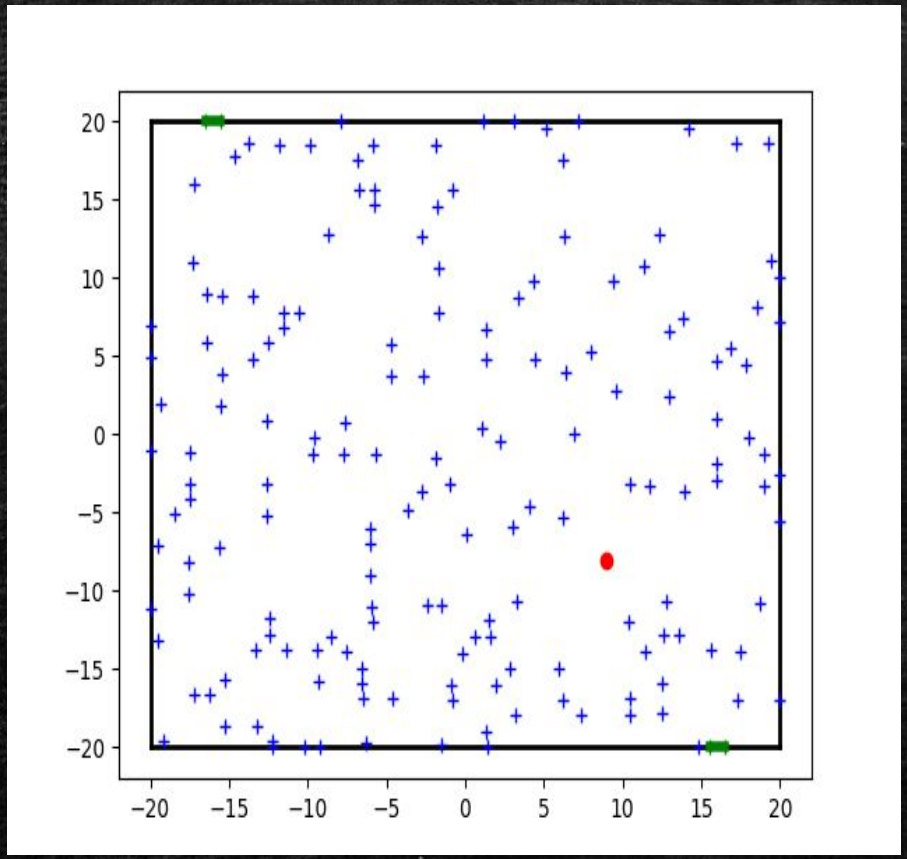
La fonction `norme` calcule la norme euclidienne d'un vecteur.

La fonction `distance` calcule la distance euclidienne entre un couple de points.

La fonction `sortie_proche` prend en entrée un individu, et une liste `sorties` et retourne le **numéro** de la sortie la plus proche de l'individu.

A l'instant initial un évènement se produit.

Idée de modélisation



+ : Individu (I)
● : Evenement (E)
_ : Porte

Phase d'évolution(1)

La situation dans l'enceinte est connue grâce à la liste `foule` et se modifie grâce à la fonction `evolution`

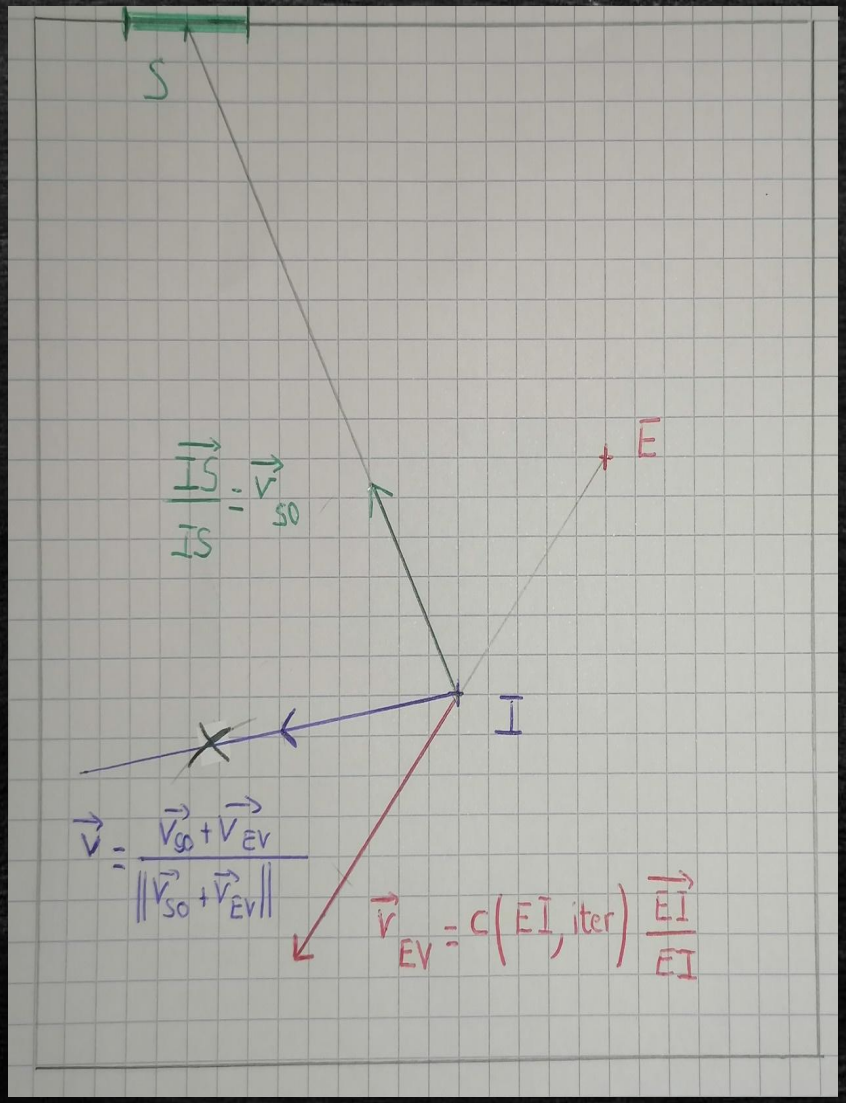
A chaque itération, cette liste est mise à jour individu par individu.

L'évolution des individus est régie par les règles suivantes.

Notons :

- I la position d'individu.
- S la sortie la plus proche de cet individu.
- E la position de l'évènement déclenchant le mouvement de panique.

Phase d'évolution(2)



Phase d'évolution(3)

On détermine un vecteur **unitaire** $\overrightarrow{v_{SO}}$ qui traduit le désir de quitter l'enceinte au plus vite :

$$\overrightarrow{v_{SO}} = \frac{\overrightarrow{IS}}{\|\overrightarrow{IS}\|}$$

On détermine un vecteur $\overrightarrow{v_{EV}}$ généré par le désir de s'éloigner au plus vite de l'évènement traumatisant survenu :

$$\overrightarrow{v_{EV}} = c(EI, \text{iter}) \frac{\overrightarrow{EI}}{\|\overrightarrow{EI}\|}$$

$$c(EI, \text{iter}) = \max\left(\frac{2N}{3EI}, 1\right) \times \frac{\text{niveau\_danger}}{\text{iter} + 1}$$

le coefficient multiplicateur $c(EI, \text{iter})$ est

- inversement proportionnel à la distance de EI
- décroissant en fonction du nombre d'itérations.

Phase d'évolution(4)

On calcule enfin $\overrightarrow{v}$ grâce à la formule suivante :

$$\overrightarrow{v} = \frac{\overrightarrow{v_{SO}} + \overrightarrow{v_{EV}}}{\|\overrightarrow{v_{SO}} + \overrightarrow{v_{EV}}\|}$$

On peut supposer $\overrightarrow{v}$ unitaire :

* D'une part, la forte densité de population rend fortement plausible le fait que les personnes se déplaceront toutes à la même vitesse.
* D'autre part, la précipitation générée par l'évènement peut être prise en compte par une diminution de la durée associée à chaque itération.

La nouvelle position de l'individu est obtenue avec

les coordonnées du point $I + \overrightarrow{v}$.

Nombre de bousculades(1)

On souhaite obtenir nb **le nombre de bousculades** engendrées par les mouvements.

- Initialement nb est nul.
- A chaque itération, est aussi calculé le nombre de nouvelles bousculades qui est ensuite ajouté à nb.
- En fin de procédure, on retourne le nombre nb.
  - Soit toutes les personnes ont atteint une sortie
  - Soit le nombre maximum d'itérations est atteint.

Extrait de code d'une fonction d'expérimentation :

```
while nb_it < nb_it_max and nb_dehors < nb_ind:
```

Nombre de bousculades(2)

Le nombre de nouvelles bousculades (soit finalement nb) est incrémenté lorsque deux personnes relativement éloignées se retrouvent trop proches à la suite de leurs mouvements.

Précisément : Extrait de code la fonction `evolution`

```
#pos_ancienne est l'ancienne position de l'individu k
#pos_nouvelle est la nouvelle position de l'individu k
if i != k:
        pos = np.array([foule[i][0], foule[i][1]])
        #pos est la position de l'individu i
        d_ancienne = distance(pos_ancienne, pos)
        d_nouvelle = distance(pos_nouvelle, pos)
        if d_nouvelle < 1 and d_ancienne >= 1:
                nouvelles_bousculades += 1
```

Une fois proches, deux personnes peuvent cheminer ensemble vers la sortie sans générer de nouvelles bousculades.

Nombre de bousculades(3)

Calcul de nb_{max} un majorant du nombre maximal de bousculades

- Le nombre d'individus présents est
  $n_{ind} = (2N+1)^2 \times \texttt{densite}$
- Le nombre maximal de bousculades par itération est donc :
  $n_{bou} = \binom{n_{ind}}{2} = \frac{n_{ind}(n_{ind}-1)}{2}$
- Si on effectue n_{ite} itérations alors
  $nb_{max} = n_{bou} \times n_{ite}$

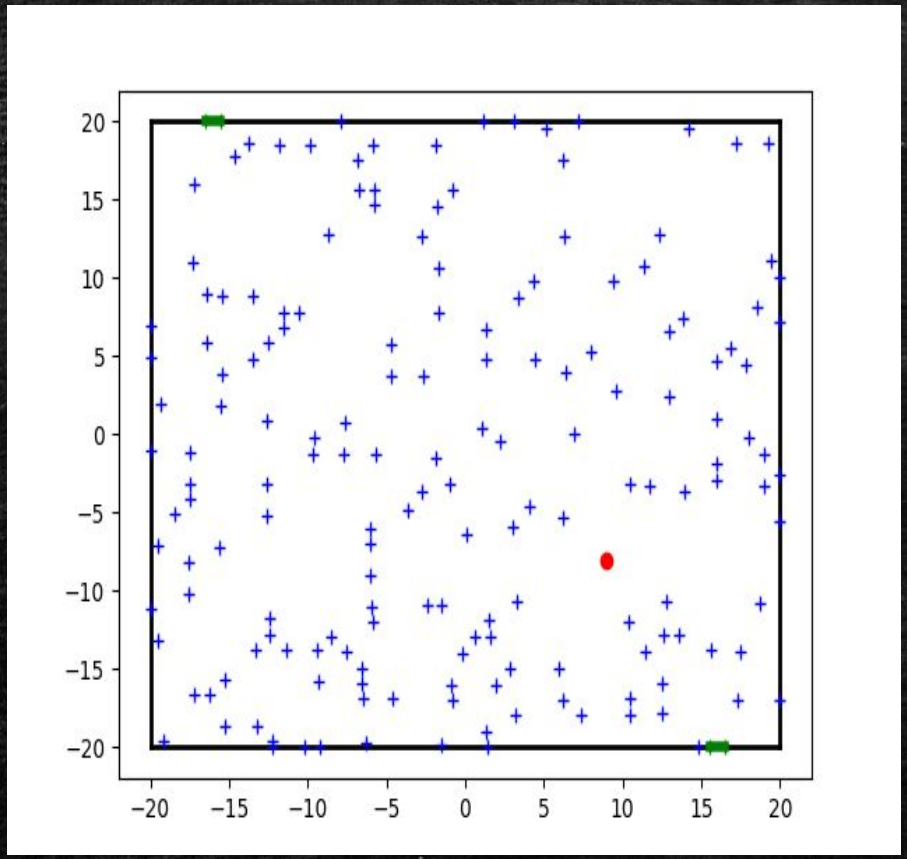
Application numérique :

Avec $N = 20$, $\texttt{densite} = 0.2$, $n_{ite} = 25$, on obtient : $nb_{max} = 1\ 407\ 837$

On peut remarquer que le nombre obtenu par simulation est considérablement inférieur à la borne nb_{max}.

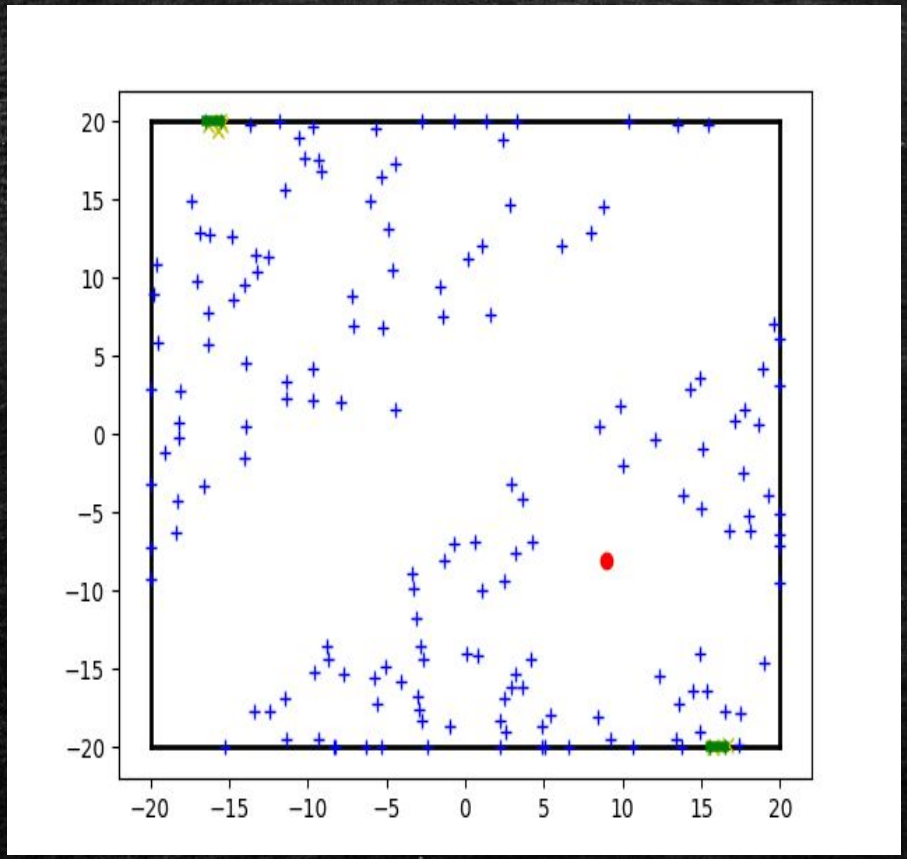
Démonstration d'une simulation

```
>>> experience(20,0.2,30,2)
```



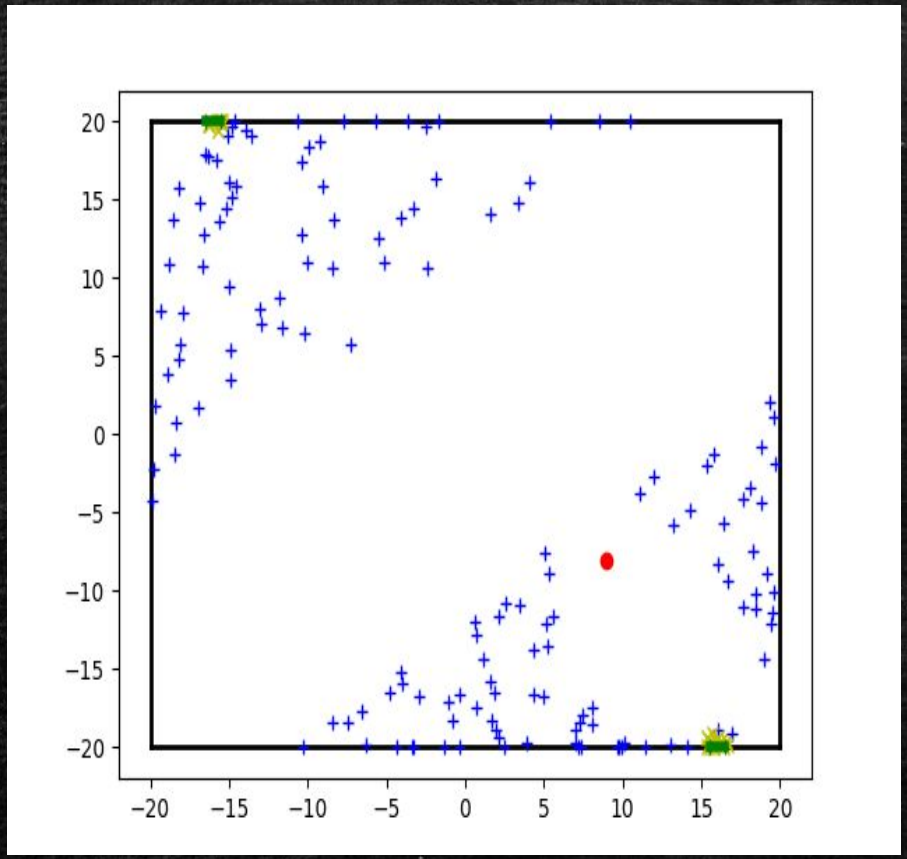
Démonstration d'une simulation

```
>>> experience(20,0.2,30,2)
```



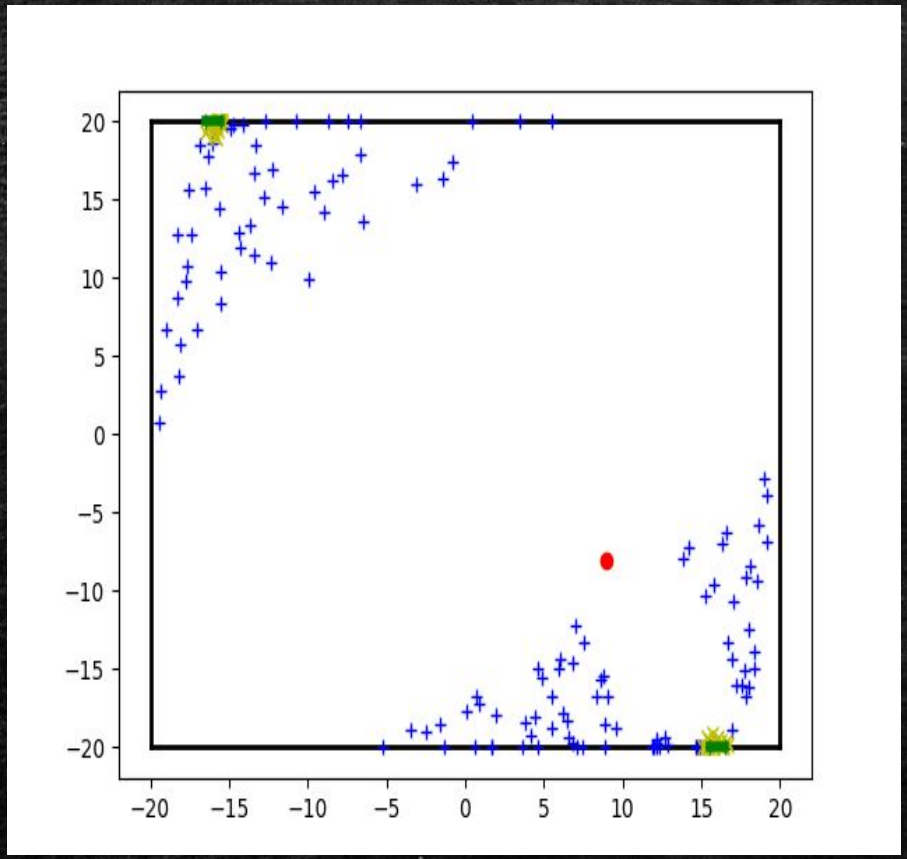
Démonstration d'une simulation

```
>>> experience(20,0.2,30,2)
```



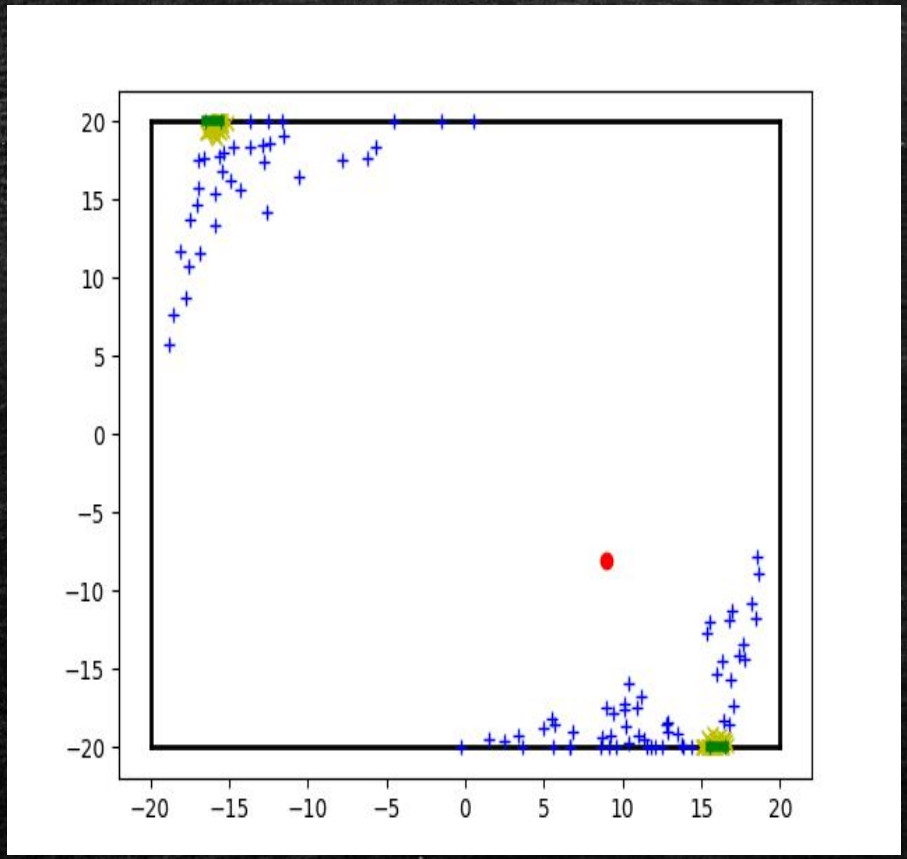
Démonstration d'une simulation

```
>>> experience(20,0.2,30,2)
```



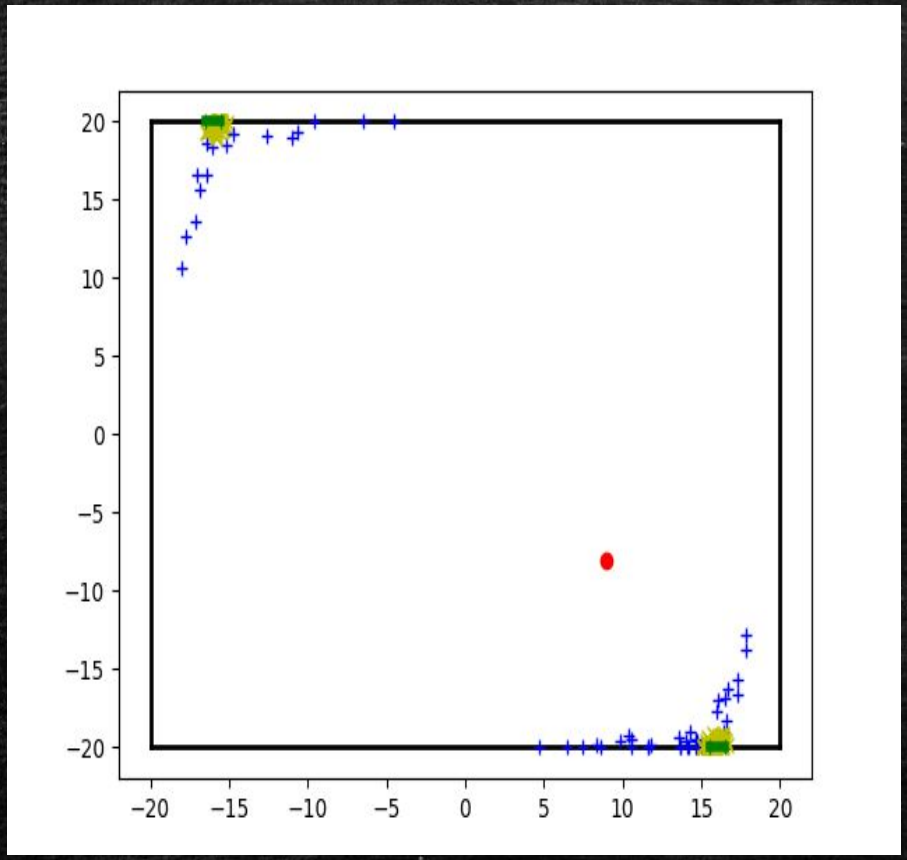
Démonstration d'une simulation

```
>>> experience(20,0.2,30,2)
```



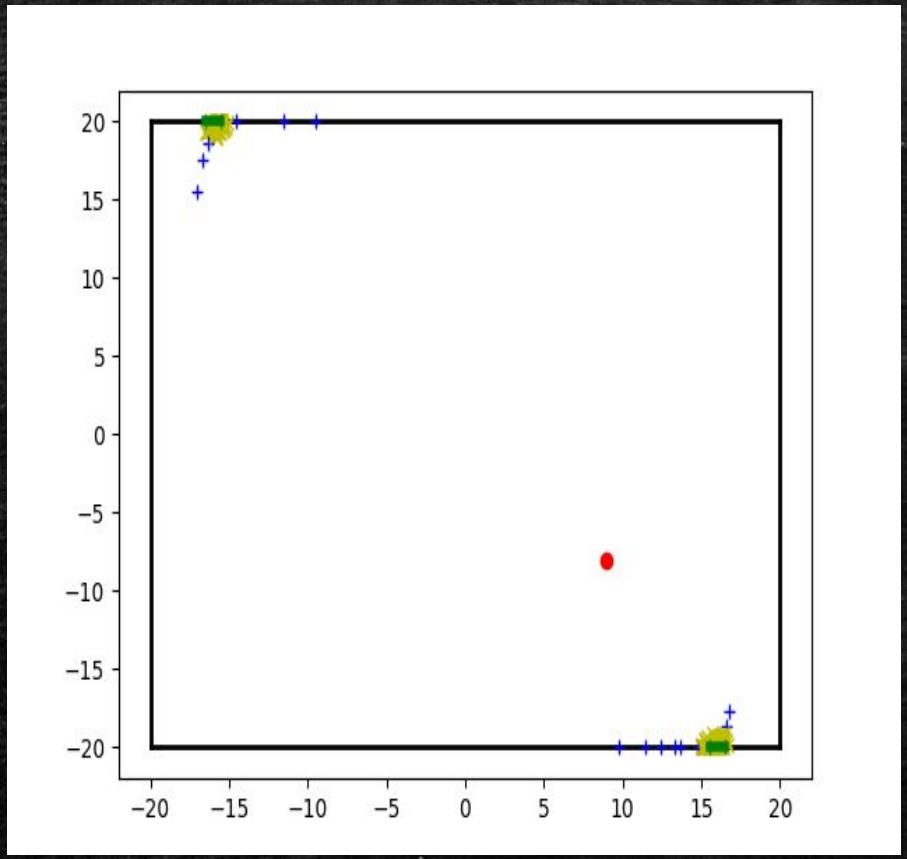
Démonstration d'une simulation

```
>>> experience(20,0.2,30,2)
```



Démonstration d'une simulation

```
>>> experience(20,0.2,30,2)
```



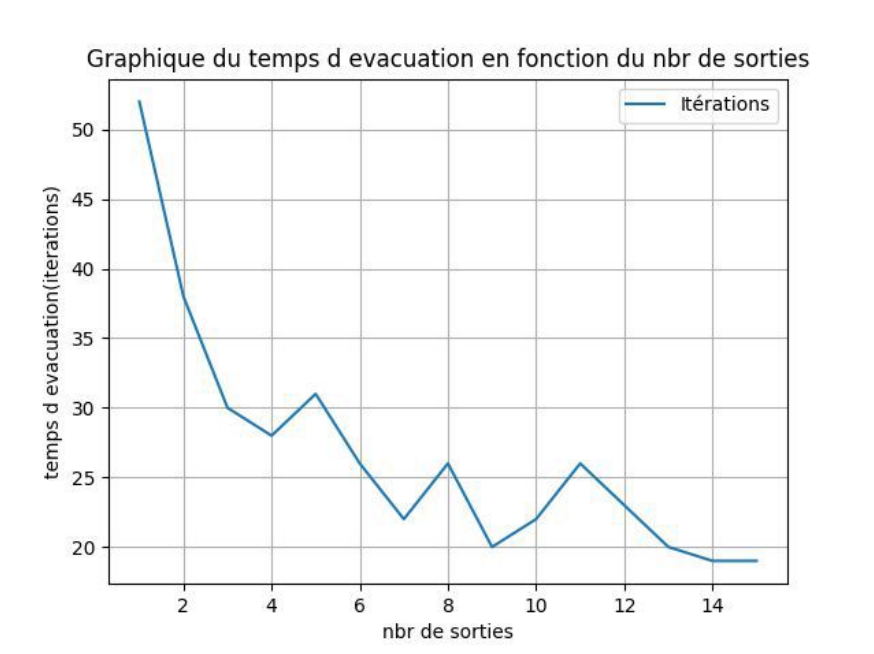
Analyse et traitement

Les résultats obtenus avec la fonction `evolution` peuvent avoir les applications suivantes :

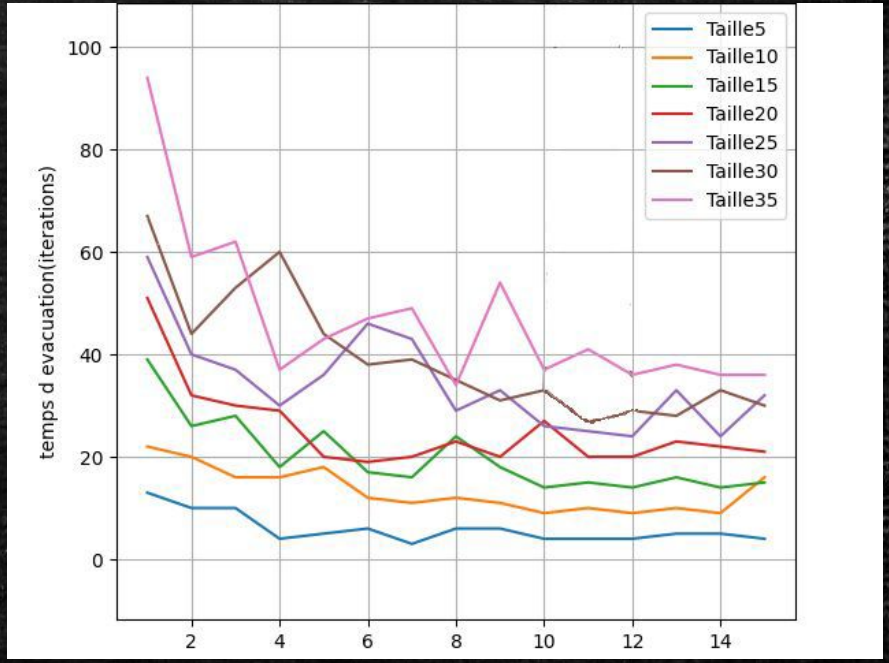
- **Estimation du temps d'évacuation** en multipliant le nombre d'itérations nécessaires pour évacuer par un temps correspondant à la durée d'une itération.
- **Estimation du nombre de blessés** corrélé au nombre de bousculades.
- **Pertinence d'augmenter le nombre de sorties** d'une salle en étudiant l'évolution du nombre de bousculades et du nombre d'itérations nécessaires pour évacuer par rapport au nombre de sorties.
- **Connaissance approximative du positionnement de la foule** à l'aide des fonctions de visualisation à un instant donné.

Graphiques Temps d'évacuation

Nombre d'itérations nécessaires pour que tous les individus soient hors de danger

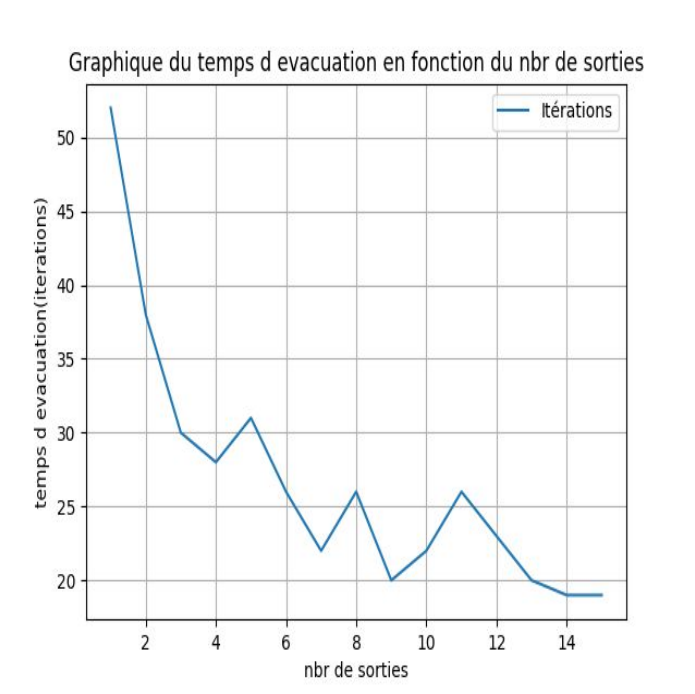


Selon la taille de la pièce on estime le nombre d'itérations pour que tous les individus soient hors de danger

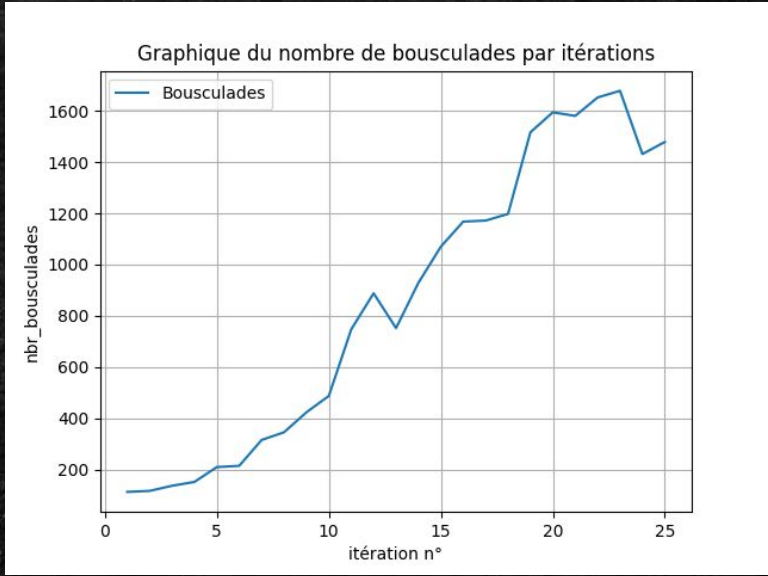


Analyse du nombre de bousculades générées dans une situation de panique

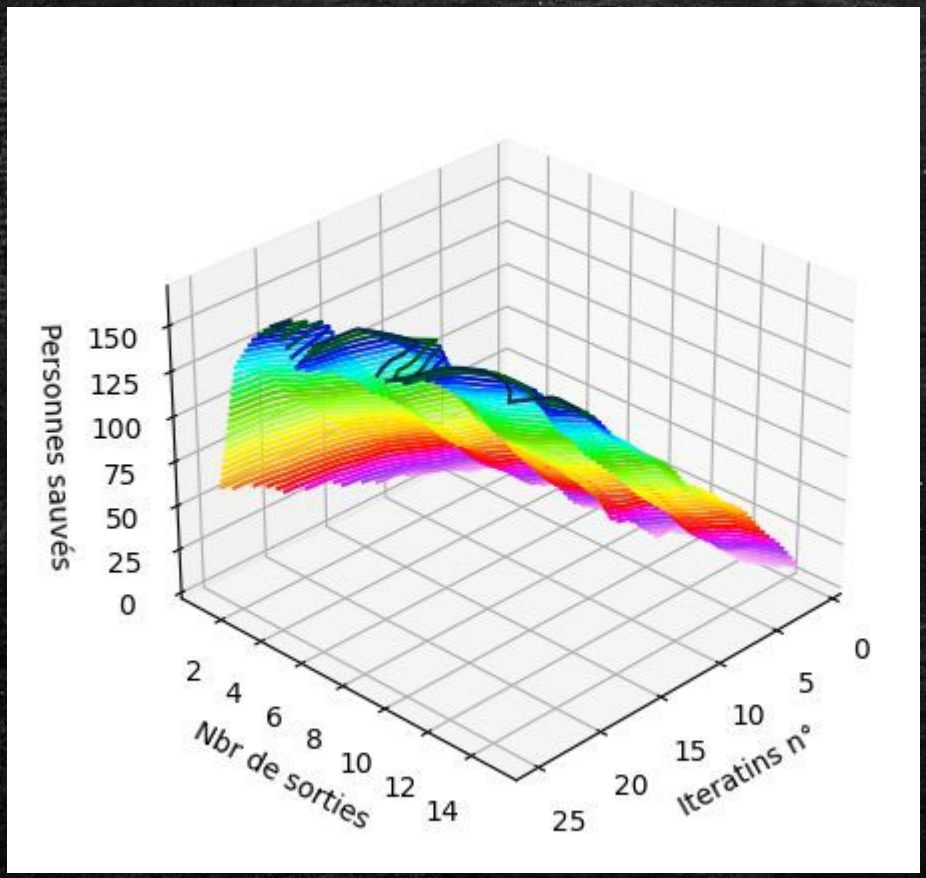
Plus le nombre de sorties est élevé, plus la sécurité est garantie

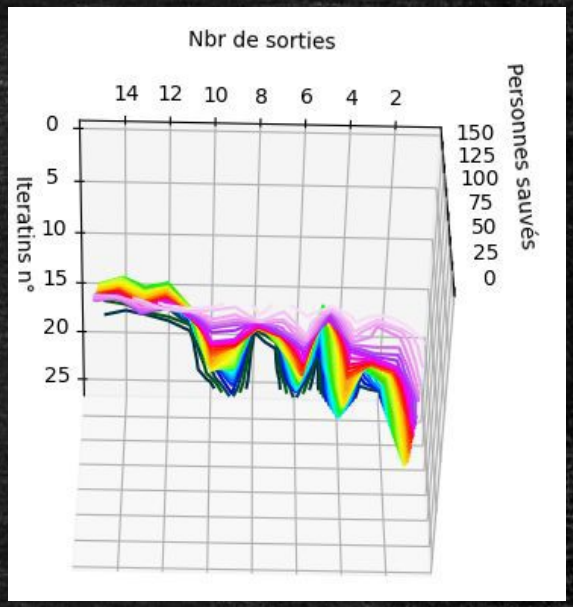


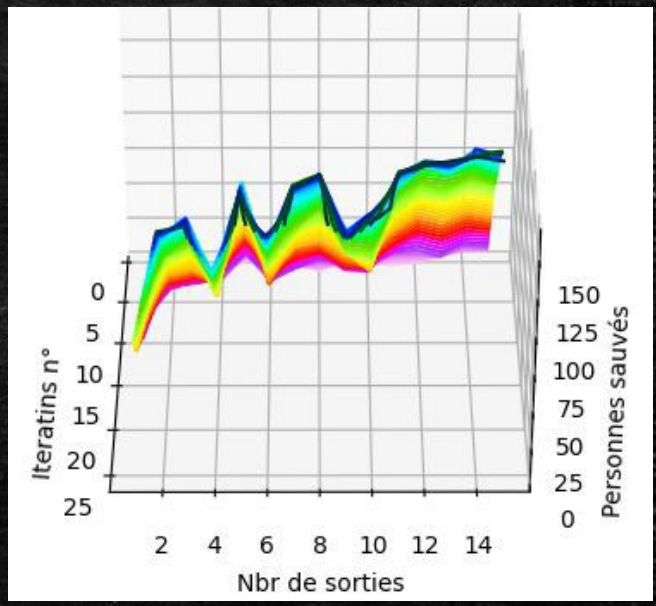
Les attroupements sont les moments où le risque est à son maximum



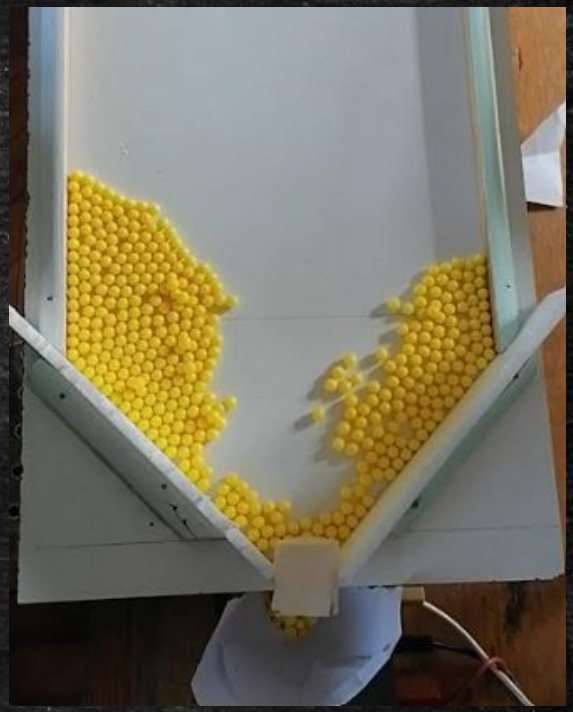
Etude 3D, Des personnes ayant quitté la zone de danger



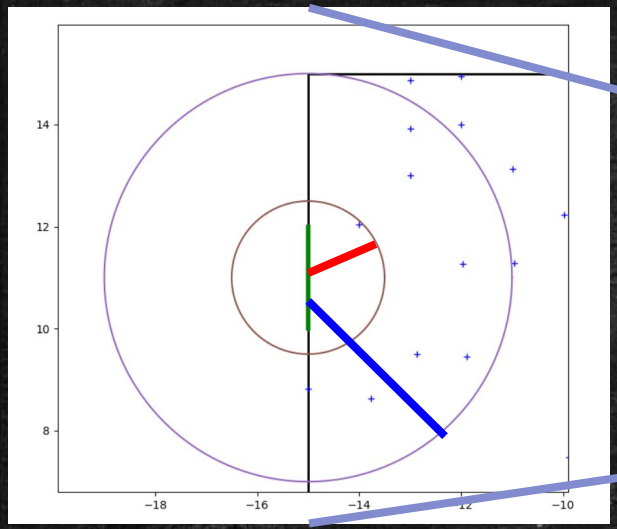


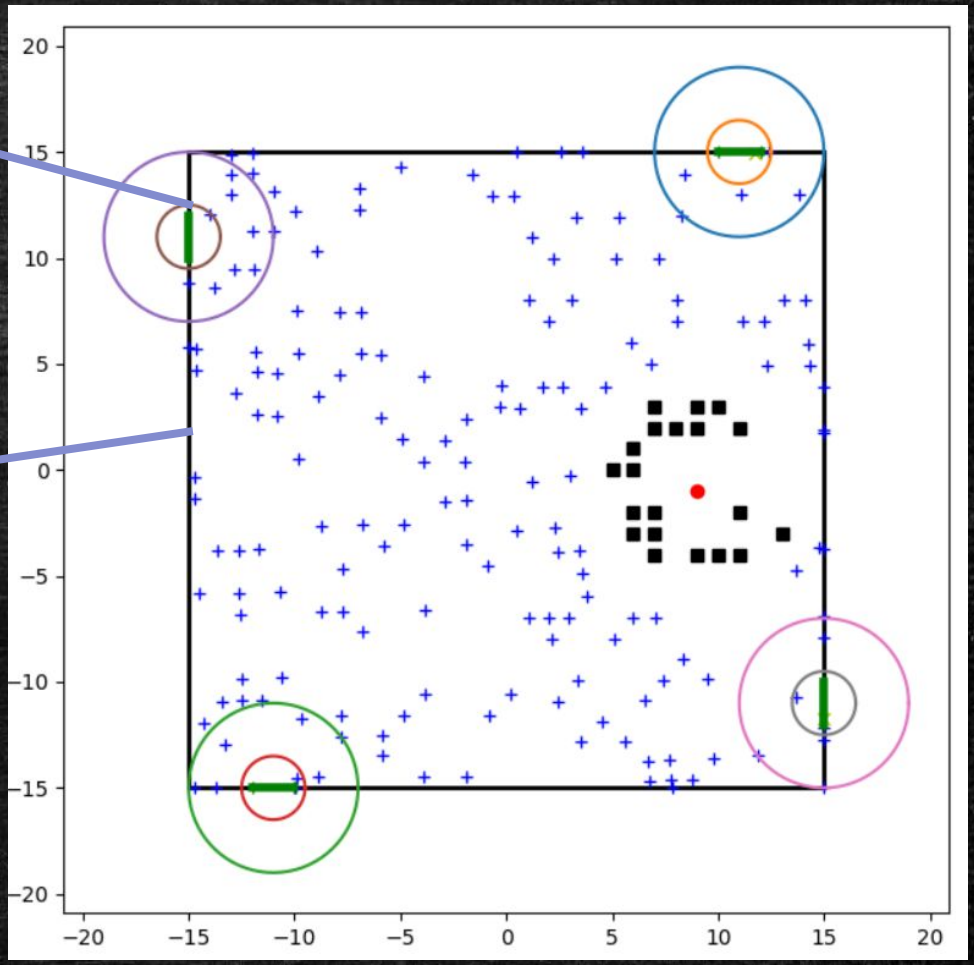


Fabrication d'une maquette



Mise en avant des effets d'arche

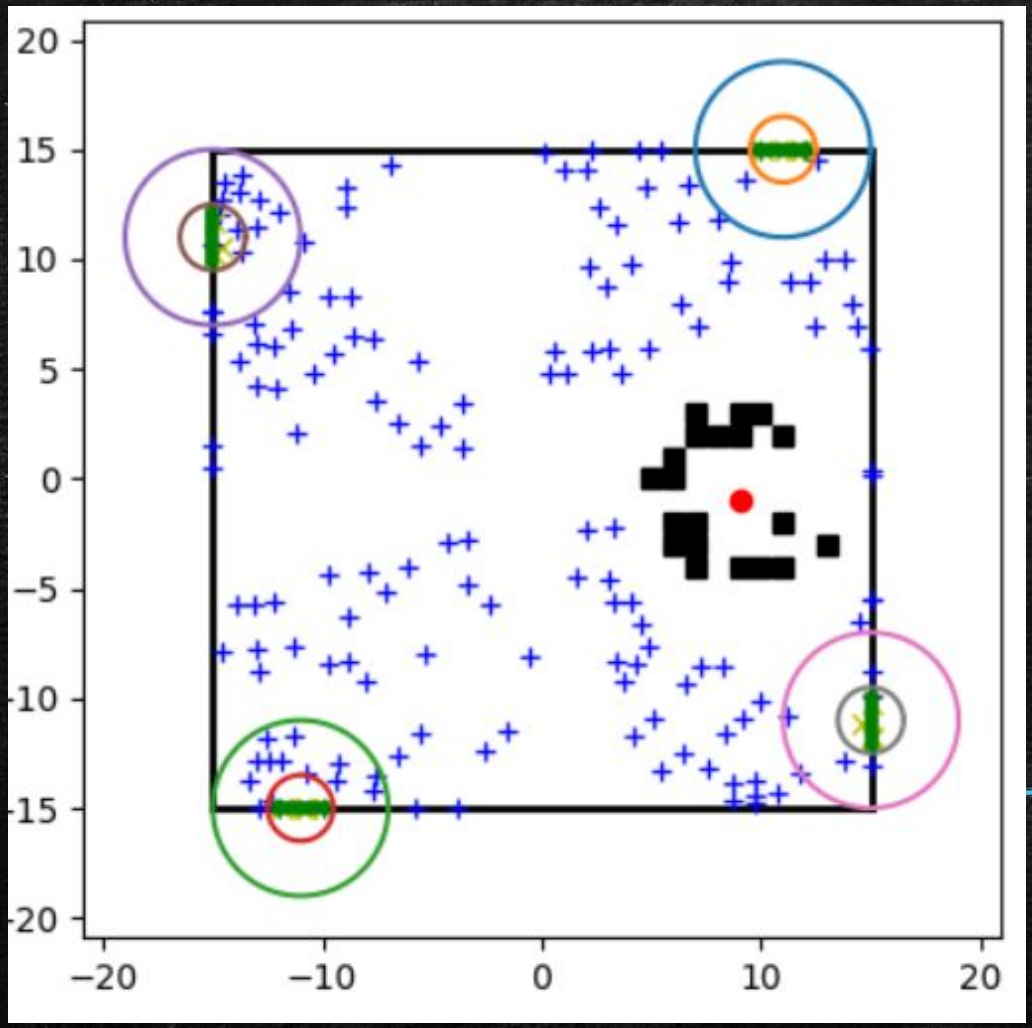


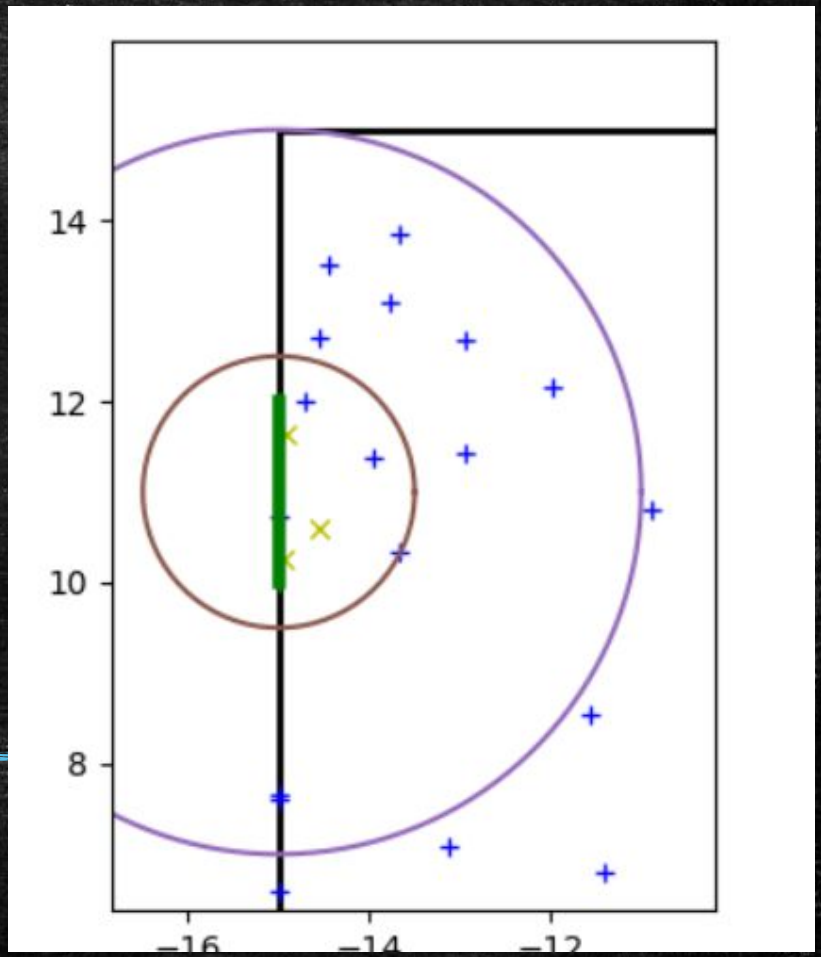


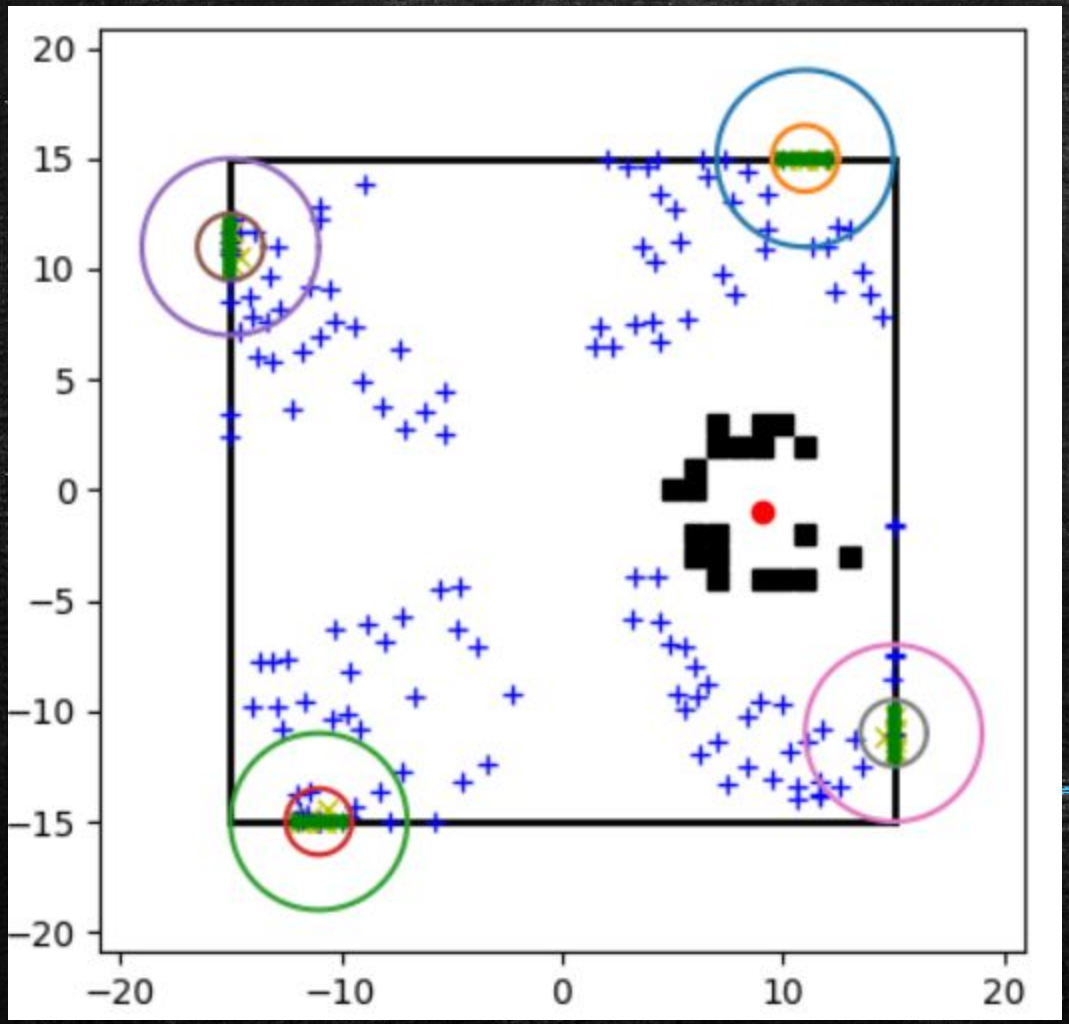
_ : rayon de blocage

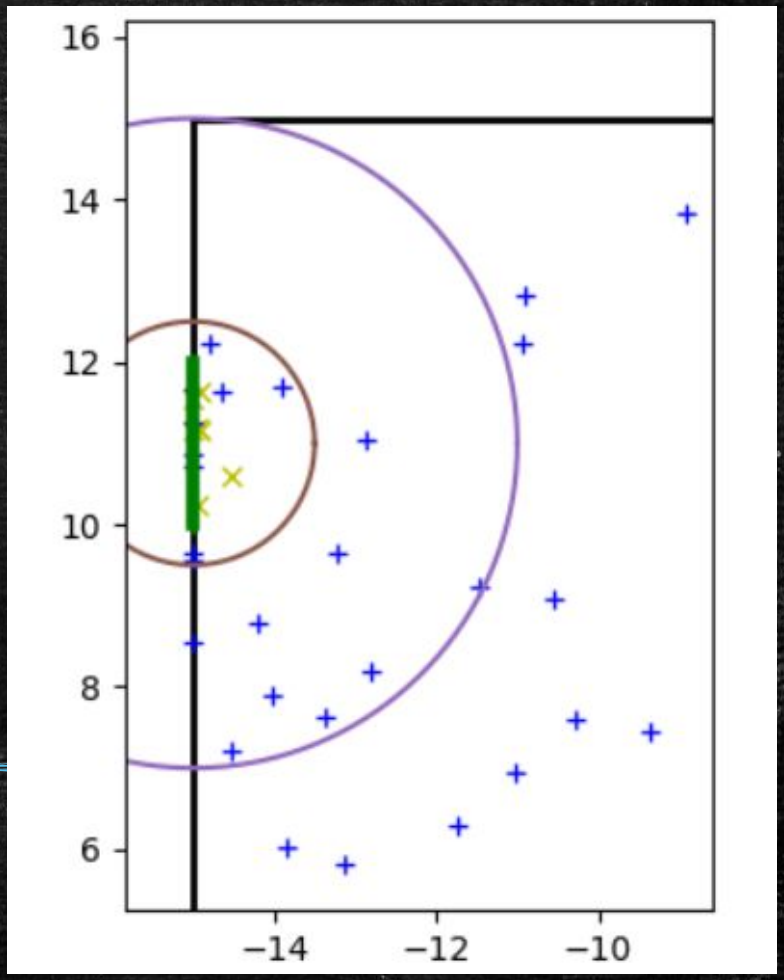
_ : rayon d'obstination

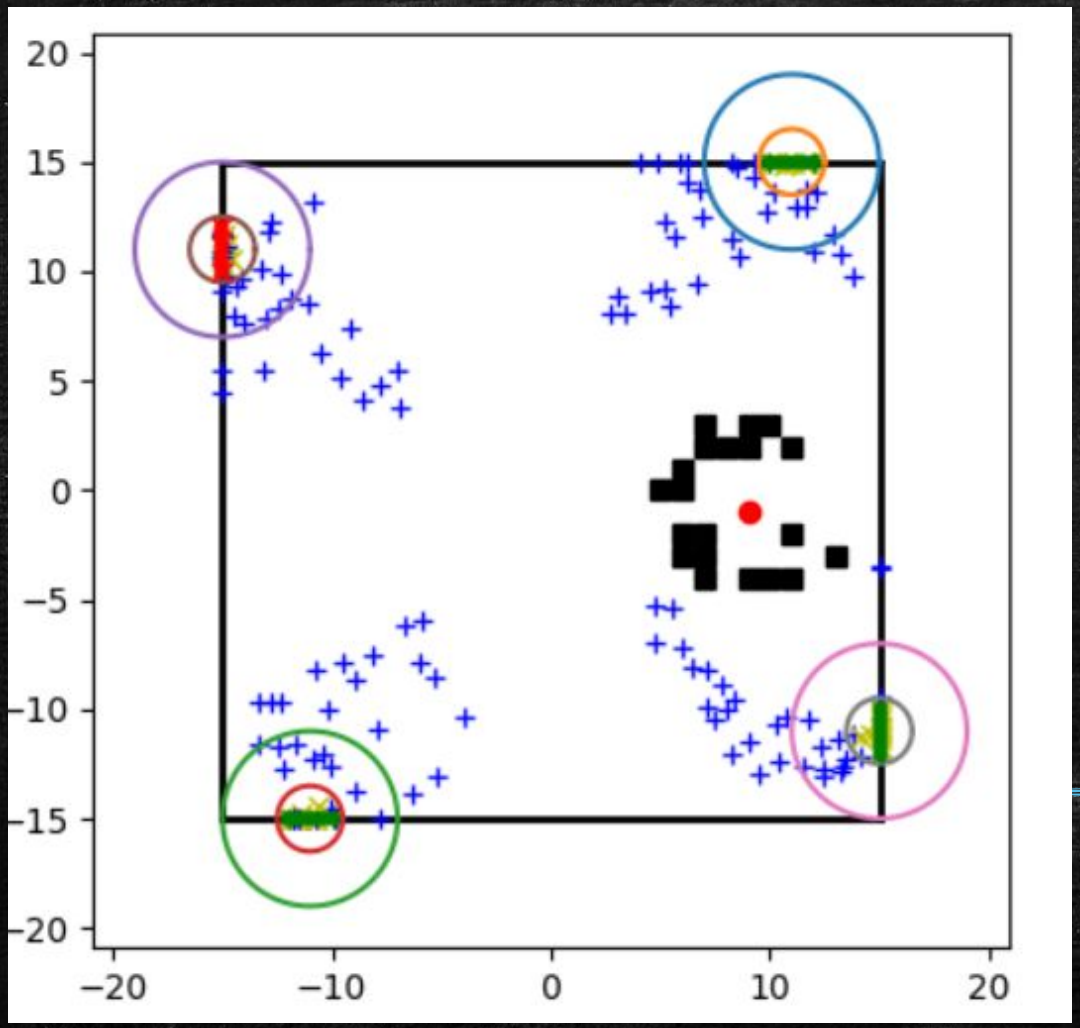
Amélioration de notre modèle en 2D

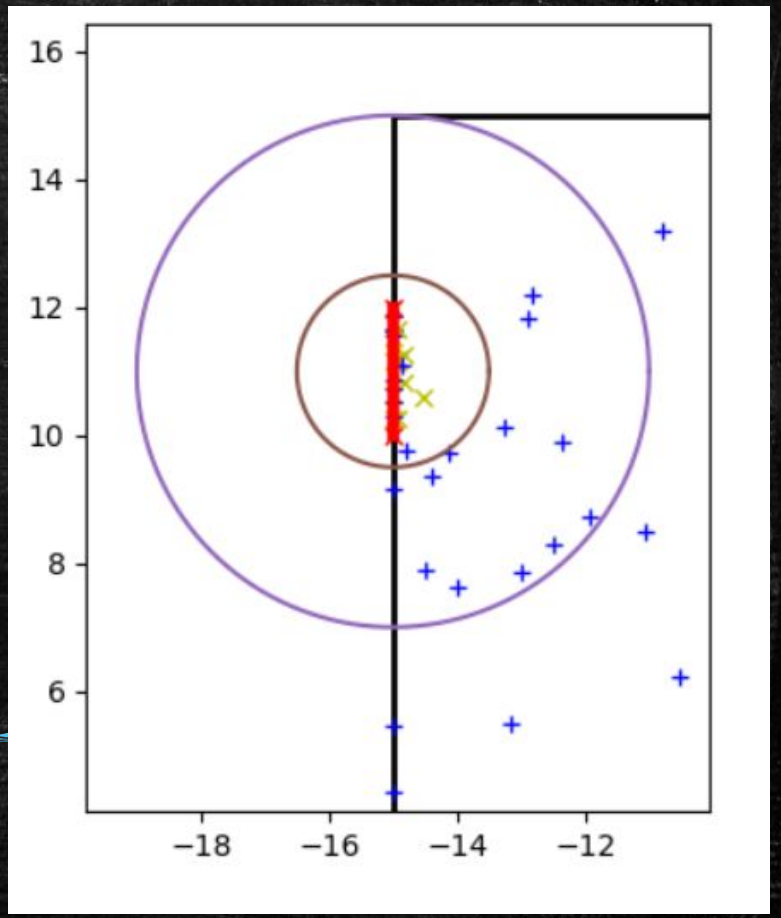

20
15
10
5
0
−5
−10
−15
−20
−20
−10
0
10
20

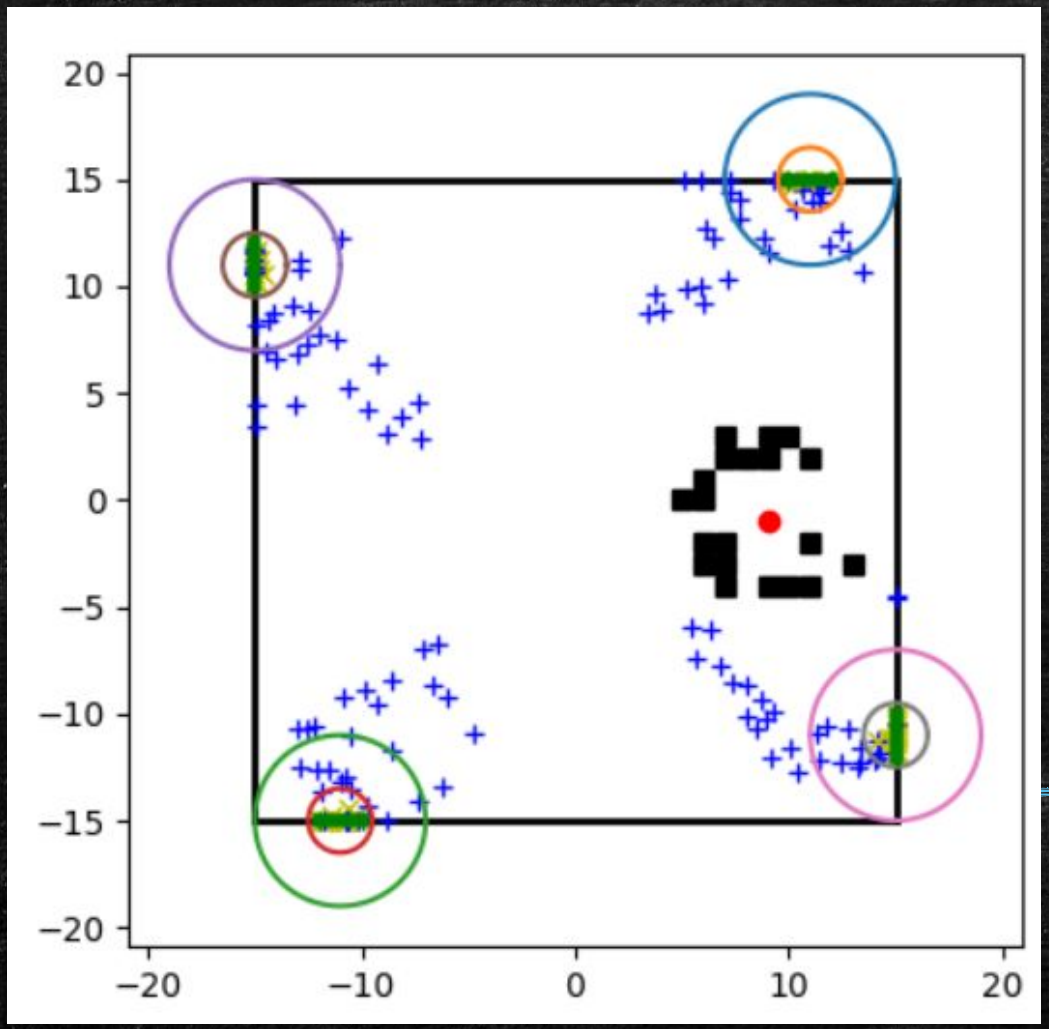

14
12
10
8
−16
−14
−12

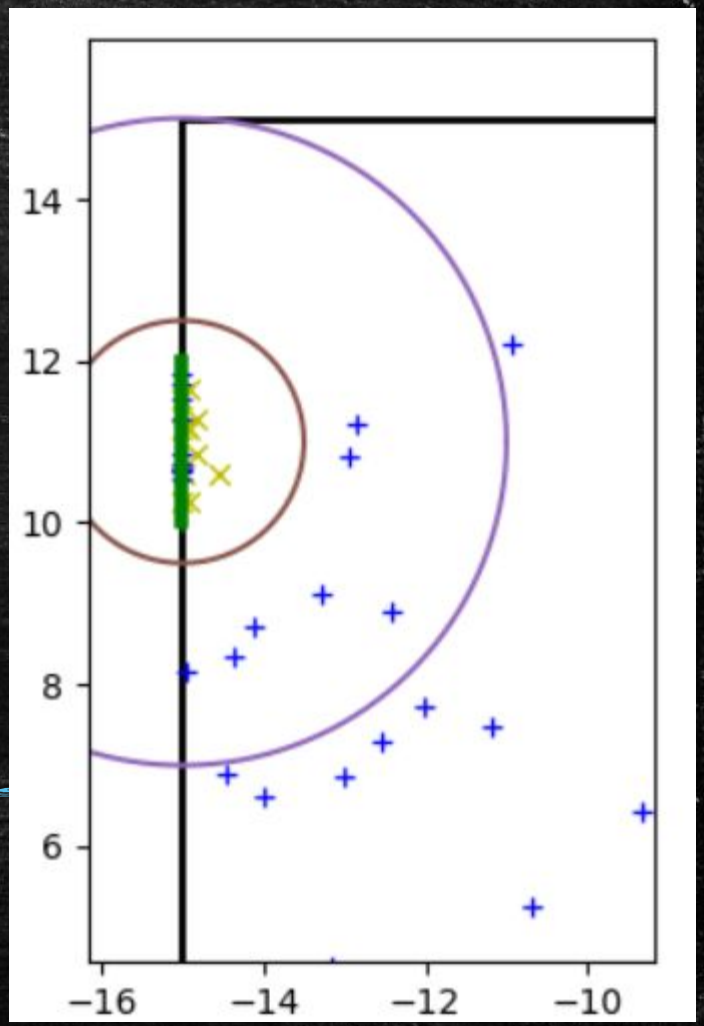

20
15
10
5
0
−5
−10
−15
−20
−20
−10
0
10
20

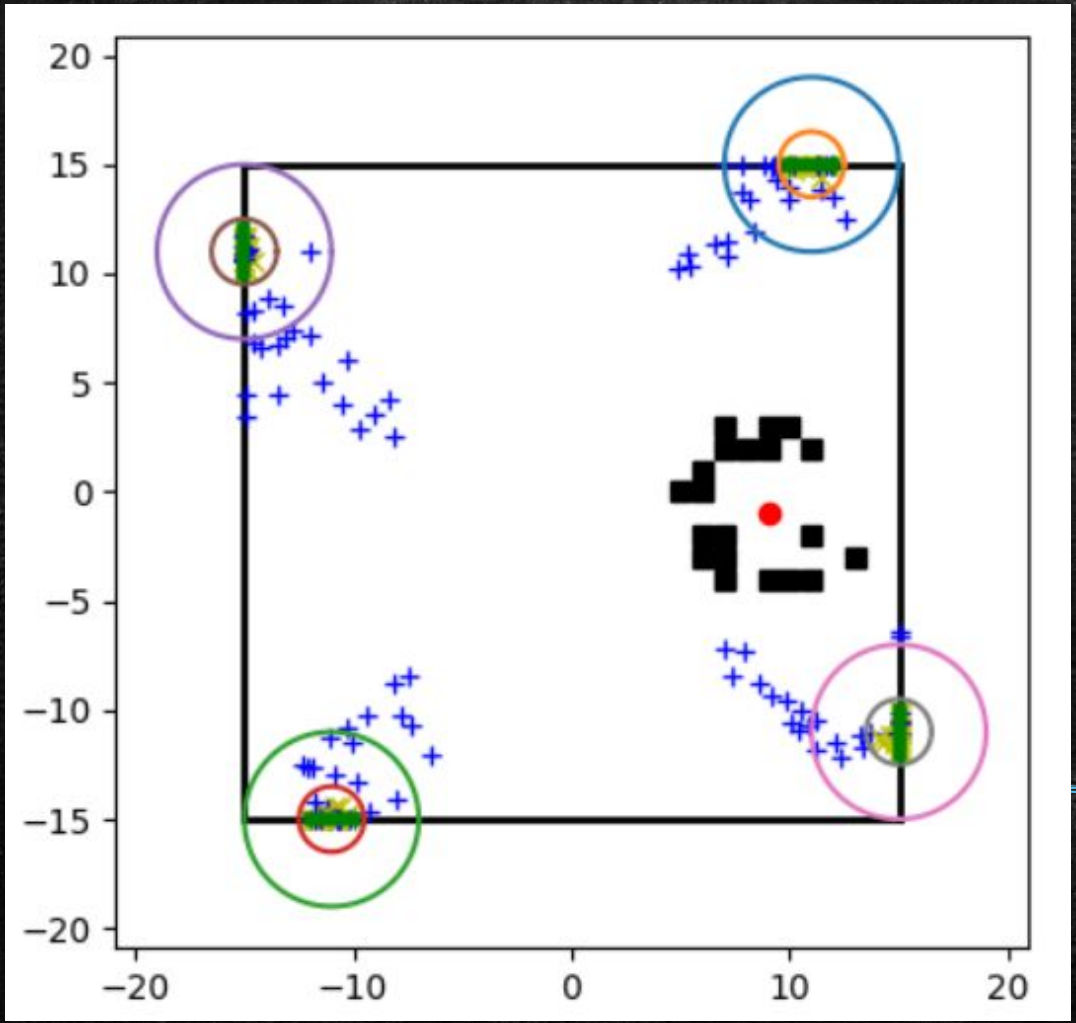

16
14
12
10
8
6
−14
−12
−10

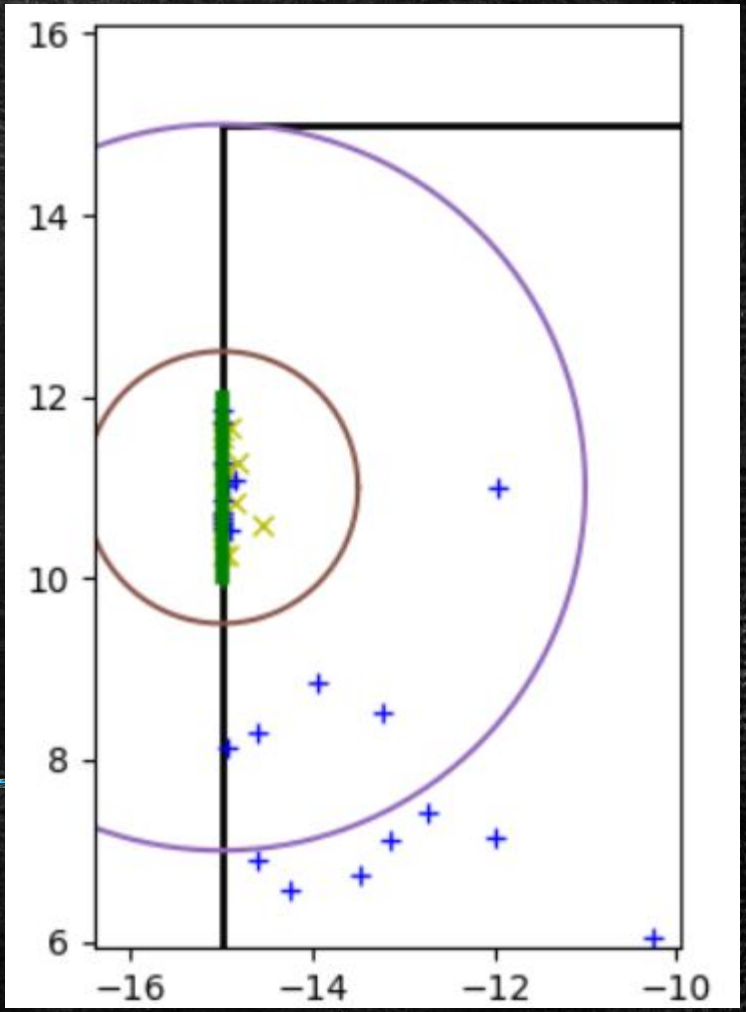

20
15
10
5
0
−5
−10
−15
−20
−20
−10
0
10
20

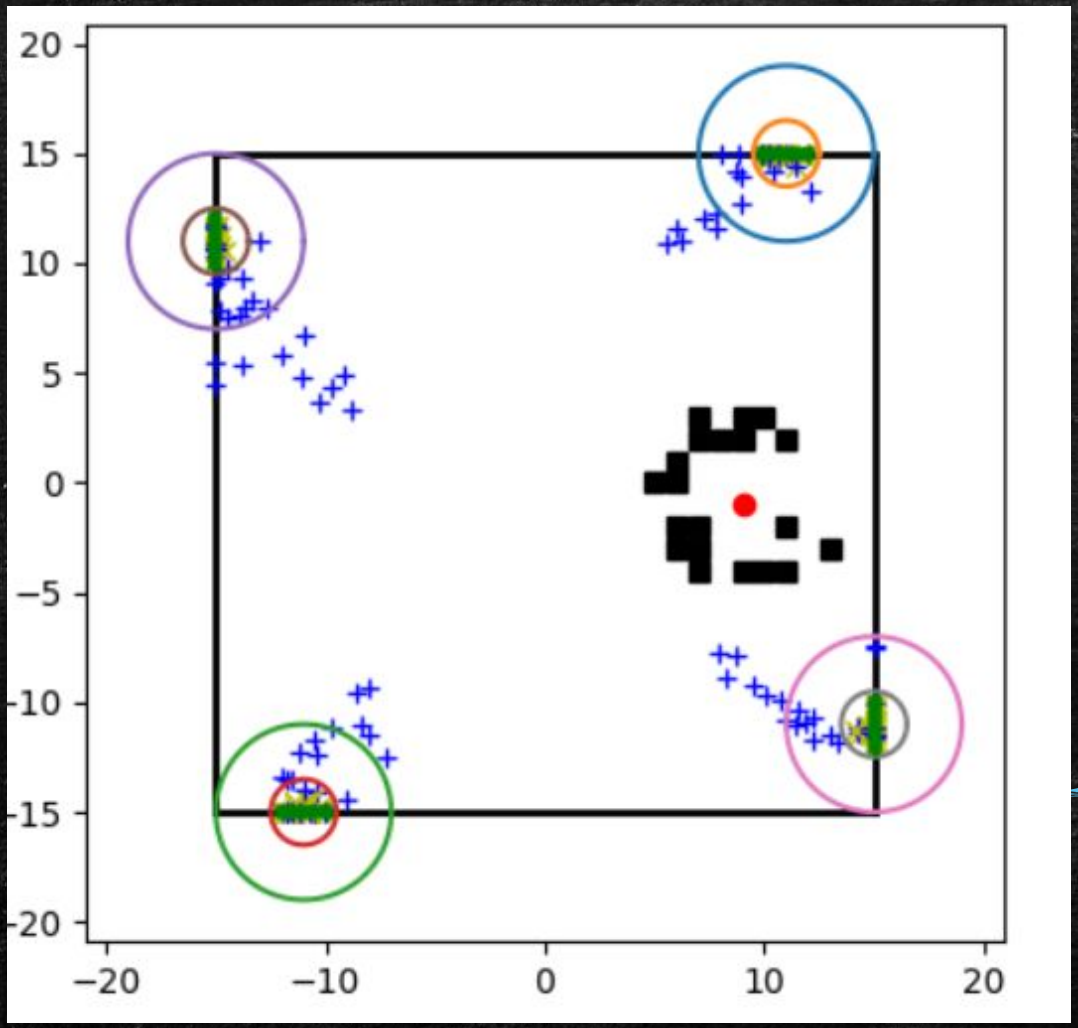

16
14
12
10
8
6
−18
−16
−14
−12

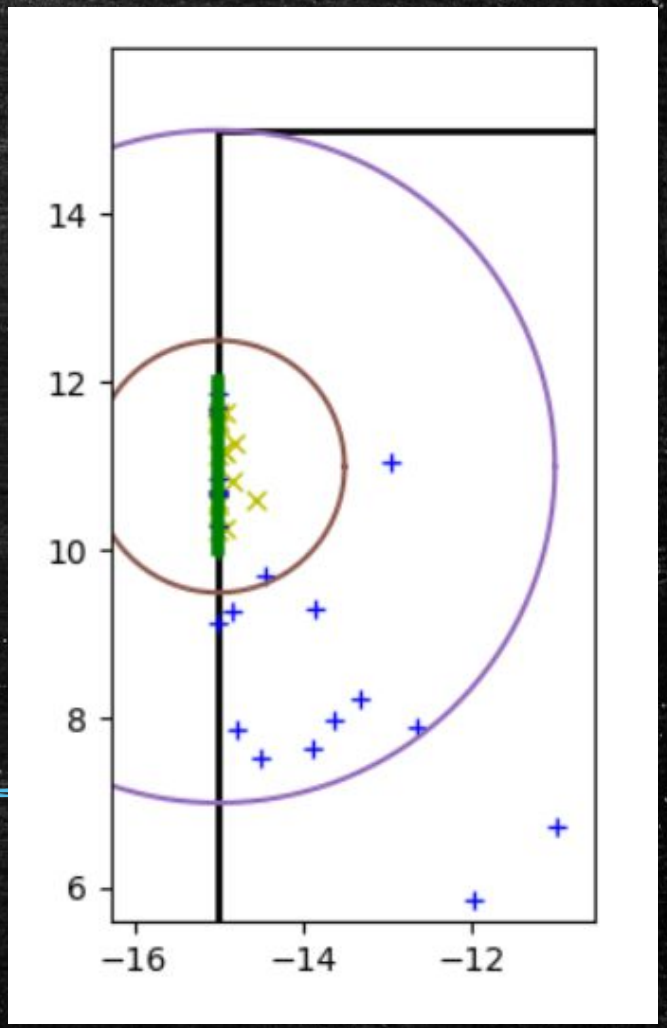

20
15
10
5
0
−5
−10
−15
−20
−20
−10
0
10
20

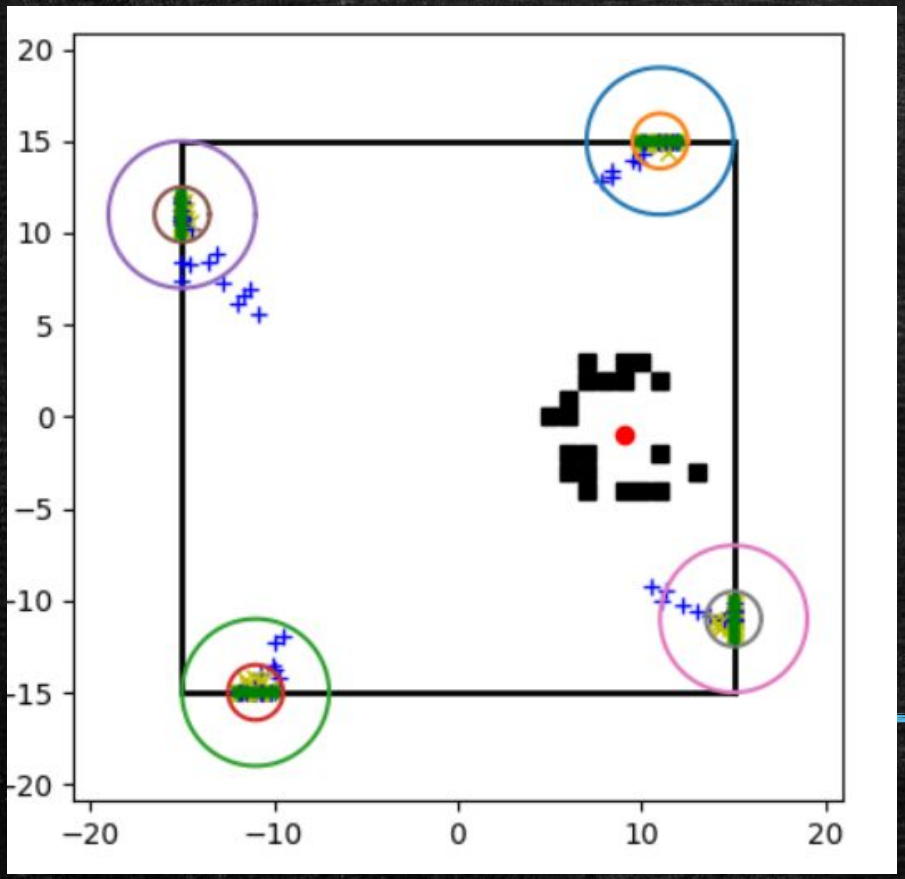

14
12
10
8
6
−16
−14
−12
−10

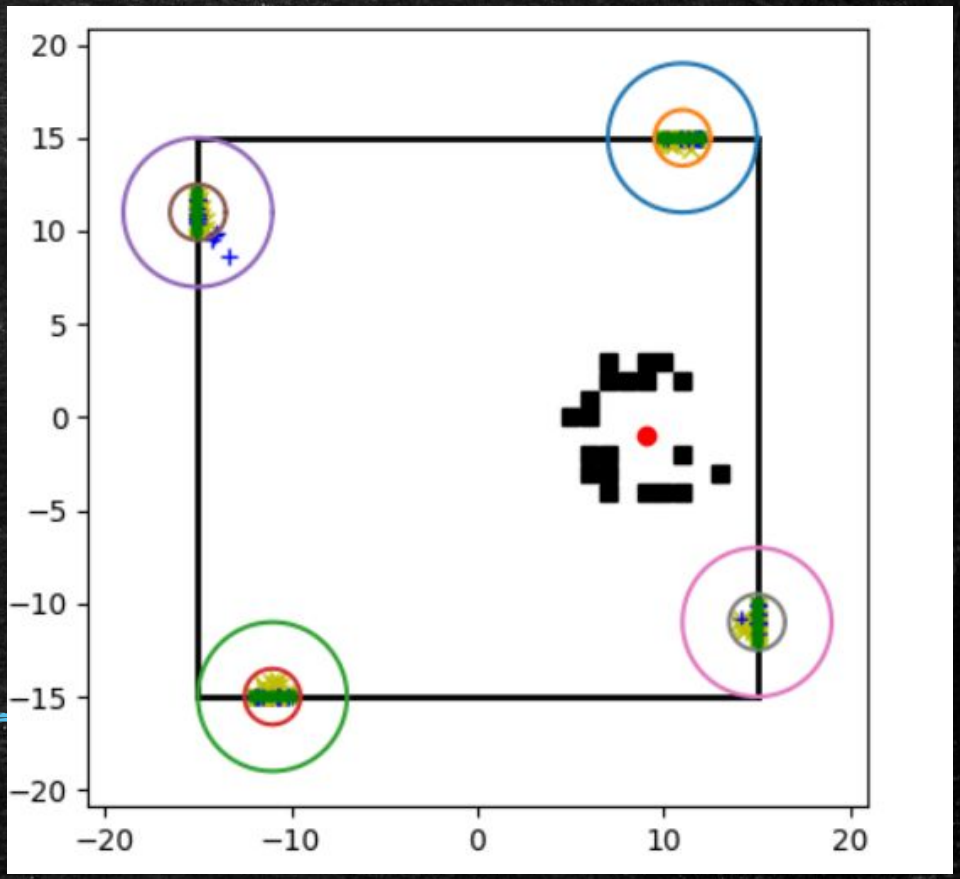

20
15
10
5
0
−5
−10
−15
−20
−20
−10
0
10
20

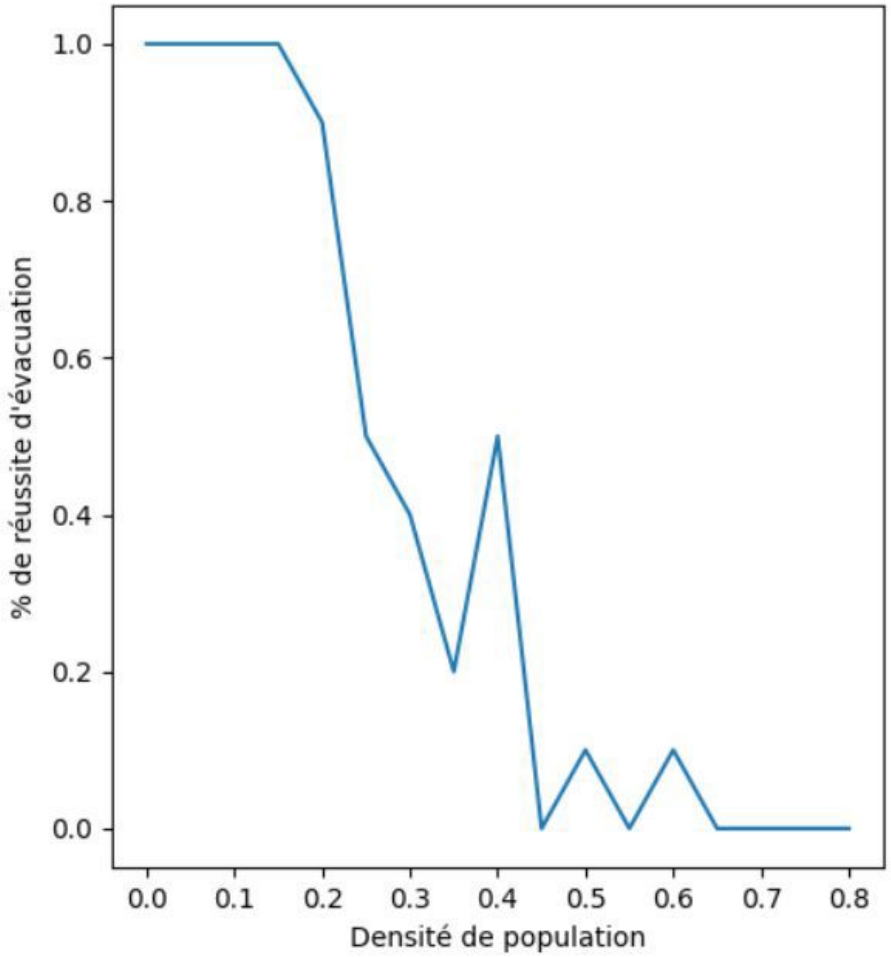

16
14
12
10
8
6
−16
−14
−12
−10

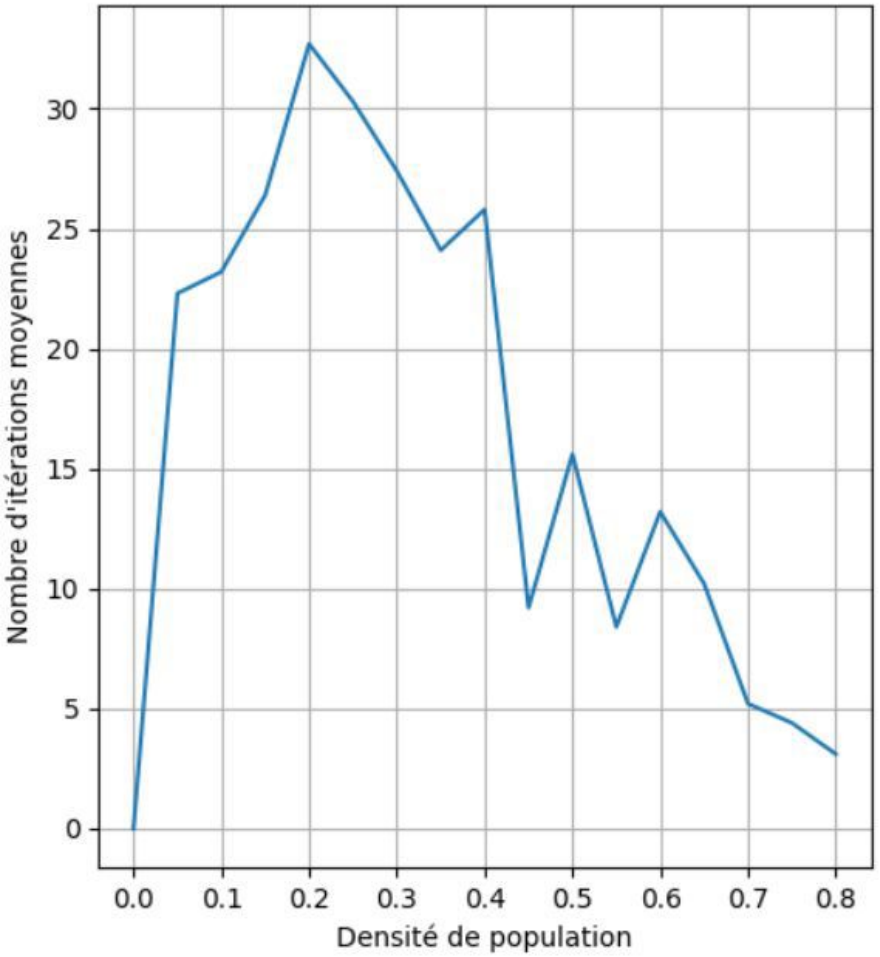

20
15
10
5
0
−5
−10
−15
−20
−20
−10
0
10
20


14
12
10
8
6
−16
−14
−12


20
15
10
5
0
−5
−10
−15
−20
−20
−10
0
10
20


20
15
10
5
0
−5
−10
−15
−20
−20
−10
0
10
20


% de réussite d'évacuation
1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
Densité de population


Nombre d'itérations moyennes
30
25
20
15
10
5
0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
Densité de population

Comparaison de nos deux modèles 2D

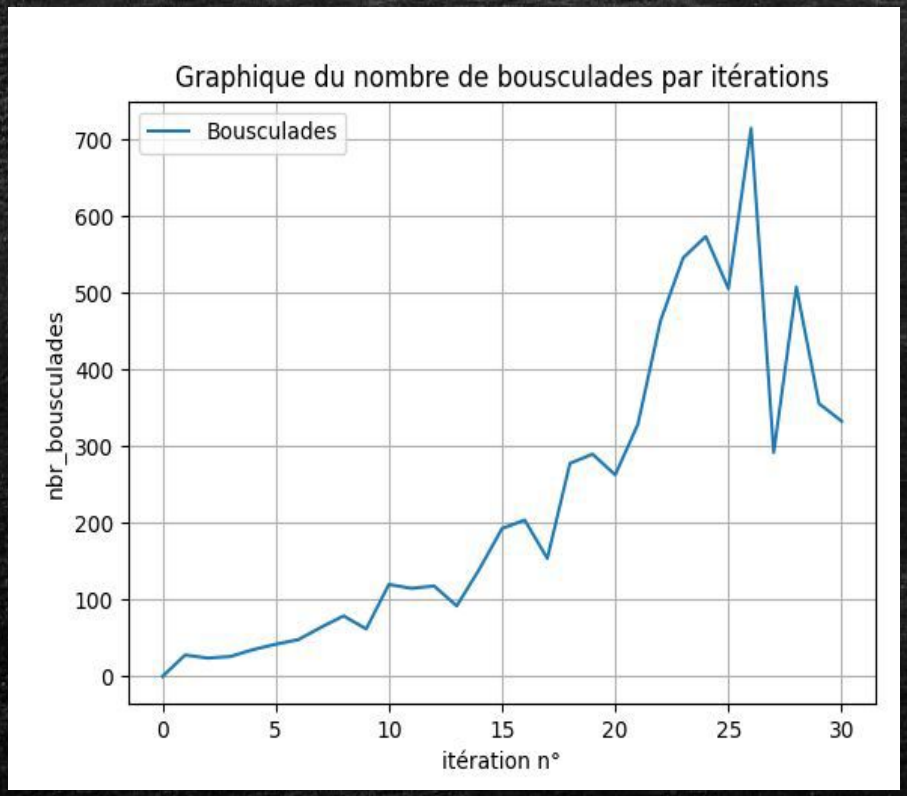
Critiques, perfectionnements et améliorations

- Qualitativement, les évolutions sont prévisibles et la réalisation de modèles parfaitement adaptés à la réalité pour obtenir des prévisions quantitatives fiables est difficile à obtenir.

  Cependant des changements de paramètres permettent d' adapter les fonctions à de nouvelles situations.
- A chaque évènement , on peut associer un coefficient augmentant ou diminuant son caractère répulsif .
- On peut créer un nouveau statut « immobilisé/blessé » pour les personnes très près du lieu de l'évènement à l'instant initial
  - ◇ A chaque individu « immobilisé/blessé », on peut associer aussi un vecteur répulsion.

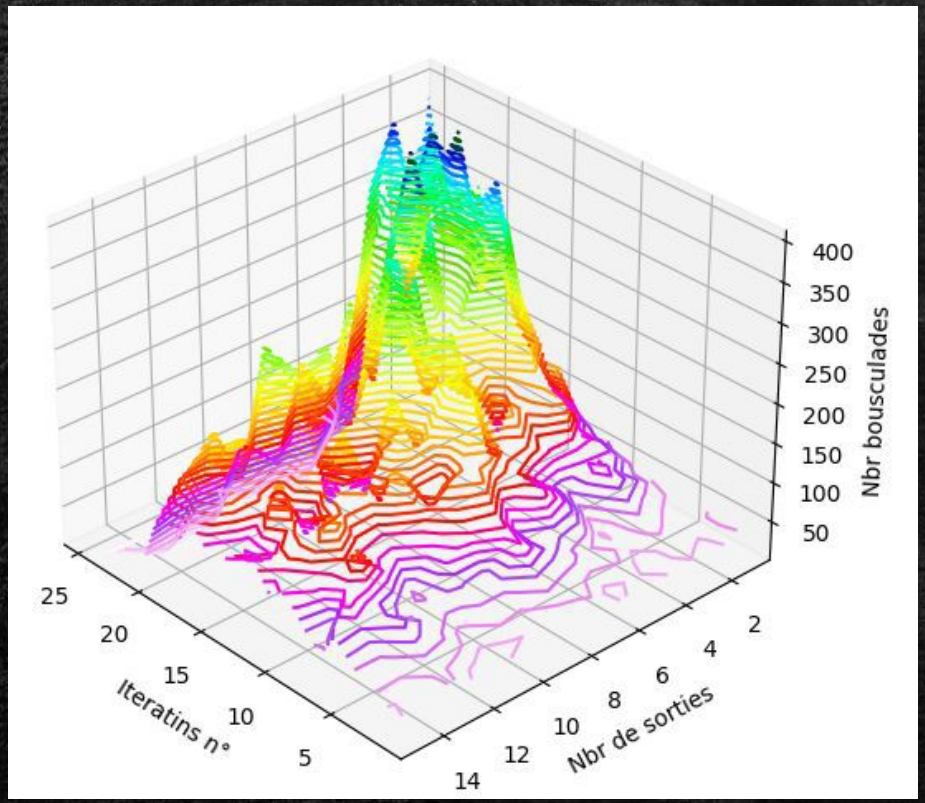
Merci pour votre écoute

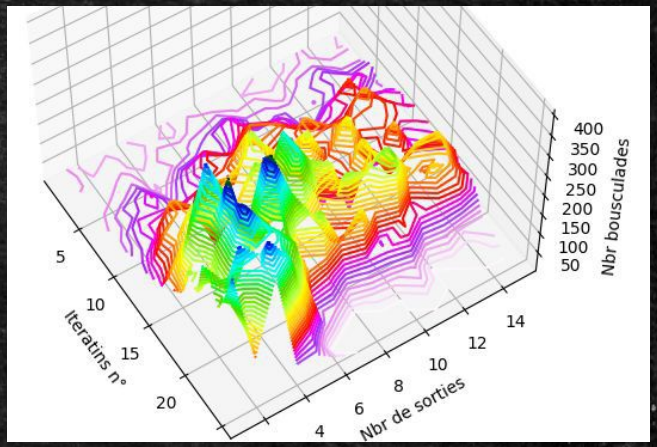
Annexe

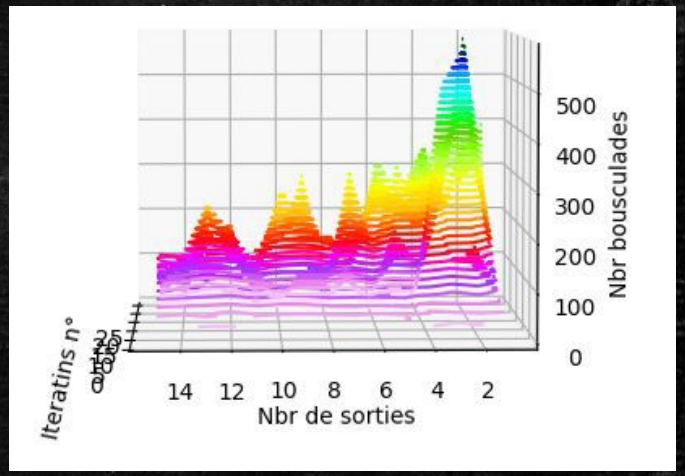


30 * 700 = 21000
Or
21000 << 1407837
Où
1407837 = nbmax

Etude 3D, Des bousculades provoquées par la panique







```python
# mouvements_foule.py

import random as rd
import numpy as np
import matplotlib.pyplot as plt
import math as m

couleurs =['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']

def creation_foule(N, densite):
    # 0 < densite < 1 ; densite = 0.2
    """individus deux à deux distincts """
    nb_ind = int ((2 * N + 1) ** 2  * densite )
    foule = []
    while len(foule) < nb_ind:
        x, y = rd.randint(-N, N), rd.randint(-N, N)
        if [x, y] not in foule:
            foule.append([x * 1., y * 1.])
    foule = np.array(foule)
    return foule


def creation_sorties(N, nbr):
    """creation d'une liste de sorties"""
    sorties = []
    while len(sorties) < nbr:
        sortie = np.zeros([2,2])
        x, y = rd.randint(-N, N), N
        if [x, y] not in sorties:
            sorties.append([x * 1., y * 1.])
            sorties.append([ -x * 1., -y * 1.] )
            sorties.append([-y * 1., x * 1.] )
            sorties.append([ y * 1., -x * 1.] )
    return np.array(sorties[:nbr])

def creation_evenement(N):
    """creation d'un évènement"""
    NN = 2 * N // 3 #pour centrer evenement
    xe = rd.randint(-NN, NN)
    ye = rd.randint(-NN, NN)
    return np.array([xe, ye])

def norme(vecteur):
    n = vecteur[0] ** 2  + vecteur[1] ** 2
    return m.sqrt(n)

def distance (A, B):
    d = (A[0] - B[0]) ** 2 + (A[1] - B[1]) ** 2
    return m.sqrt(d)

"""
 ef[i] = 0 ssi individu i immobilisé definitivement ;
 ef[i] = 1 ssi individu i en mouvement ;
 ef[i] = 2 ssi individu i sorti de l'enceinte
"""
```

```python
def initialisation_etat_foule(N, foule, evenement, danger):
    """ 0 ssi trop proche ; 1 sinon"""
    nb_ind = len(foule)
    nb_imm = 0
    ef = [1] * nb_ind
    r = N / 10 * danger #
    for i in range(nb_ind):
        if distance(foule[i], evenement) < r:
            ef[i] = 0
            nb_imm += 1
    return ef, nb_imm


def sortie_proche(N, individu, sorties):
    """ indice de la sortie la plus proche de individu"""
    ind_s = -1
    dist_min = 4 * N # > max sqrt2 * 2 * N
    for j in range(len(sorties)):
        dist = distance(individu, sorties[j])
        if dist < dist_min:
            ind_s = j
            dist_min = dist
    return ind_s

# def classement_sorties2(N, individu, sorties):
#     """ liste des sorties classée par proximité croissante pour individu """
#     """ A perfectionner tri seclion echange (petit nombre de sorties)
#         probleme cas egalité mal gérés """
#     distances_sorties = [distance(individu, s) for s in sorties]
#     ds = sorted(distances_sorties)
#     so = []
#     for d in ds:
#         ind_s = distances_sorties.index(d)
#         so.append(ind_s)
#     return so


def classement_sorties(N, individu, sorties):
    """ tri seclion echange (petit nombre de sorties) """
    nb_sor = len(sorties)
    indices_sor = [k for k in range(nb_sor)]
    for j in range(nb_sor - 1):
        sor = [ sorties[k] for k in indices_sor[j:]]
        #print(sor, indices_sor)
        jmin = sortie_proche(N, individu, sor) + j
        #print(jmin)
        aux = indices_sor[j]
        indices_sor[j] = jmin
        indices_sor[jmin] = aux
    return indices_sor


```

```
106|
107| def visualisation(N, foule, ef, evenement, sorties, es,
taille_porte, densite_bloq):
108|     """visualisation du contour"""
109|     rayon_bloq = taille_porte
110|     nb_bloq = 1/2 * 3.14 * rayon_bloq ** 2 * densite_bloq
111|     plt.plot([-N,N],[-N,-N], 'k-', linewidth=2.0)
112|     plt.plot([-N,N],[N,N], 'k-', linewidth=2.0)
113|     plt.plot([-N,-N],[-N,N], 'k-', linewidth=2.0)
114|     plt.plot([N,N],[-N,N], 'k-', linewidth=2.0)
115|     """visualisation de la foule """
116|     for i in range(len(foule)):
117|         x, y = foule[i]  #[foule[i][0]],[foule[i][1]]
118|         if ef[i] == 0:
119|             plt.plot([x], [y], color = 'k', marker = "s")
120|             """rond rouge pour les immobilisés"""
121|         if  ef[i] == 1:
122|             plt.plot([x], [y], color = 'b', marker = "+")
123|             """croix + bleue pour les presents"""
124|         # if ef[i] == 2:
125|         #     plt.plot([x], [y], color = 'y', marker = "x")
126|         #     """croix x jaune pour les sortis """
127|     plt.plot(evenement[0], evenement[1] , color = 'r', marker =
"o" )
128|     """rond rouge pour l'évènement """
129|     for j in range(len(sorties)):
130|         xs, ys = sorties[j]
131|         ecart =  taille_porte / 2
132|         if es[j] < nb_bloq:
133|             marqueur = 'g|-' #ligne verte pour les portes
ouvertes 'g|-'
134|         else:
135|             marqueur = 'rx-' # deux croix rouges pour less
portes fermées
136|         if abs(ys) == N:
137|             #lst_coord = [xs - 0.5, xs + 0.5],[ys, ys]
138|             plt.plot([xs - ecart, xs + ecart],[ys, ys] ,
marqueur, linewidth = 4.0 )
139|         else:
140|             plt.plot([xs , xs ],[ys - ecart, ys + ecart],
marqueur, linewidth = 4.0)
141|         #plt.plot(lst_coord , marqueur, linewidth = 4.0)
142|     plt.show(block = True)
143|
144| def sortie_valide(N, nb_ind, individu, sorties, es,
taille_porte, densite_bloq):
145|     """ -1 si aucune sortie disponible """
146|     nst = 0 # nombre de sorties testées
147|     so = classement_sorties(N, individu, sorties)
148|     rayon_bloq = taille_porte
149|     nb_bloq = 1/2 * 3.14 * rayon_bloq ** 2 * densite_bloq
150|     rayon_obstination = 0.9 * rayon_bloq  ## choix important ##
151|     while nst < len(sorties):
152|         ind_s = so[nst]
153|         d = distance(individu, sorties[ind_s])
```

```
154|             if es[ind_s] < nb_bloq or d < rayon_obstination: # ==
seuil <  = <
155|                 # s'obstine si sortie proche; à améliorer
156|                 return ind_s
157|             nst += 1
158|         return -1
159|
160|
161|     def vecteur_sortie(individu, ps):
162|         """vecteur naturel pour sortir au plus vite"""
163|         vecteur = ps - individu
164|         nor = norme(vecteur)
165|         if nor == 0:
166|             return 1 / norme(ps) * ps
167|             # correction si sur sortie pour eviter eloignement du à
v_eve
168|         else:
169|             return 1 / nor  * vecteur
170|
171|
172|     def vecteur_evenement(individu, evenement, N):
173|         """ vecteur perturbation lié à l'évènement"""
174|         vec_eve = np.zeros(2)
175|         x, y = individu[0], individu[1]
176|         xe, ye = evenement[0], evenement[1]
177|         vec_eve[0] = x - xe
178|         vec_eve[1] = y - ye
179|         nor_eve = norme(vec_eve)
180|         if nor_eve <  0.0001:
181|             nor_eve = 0.0001
182|         distance_moyenne = (2 * N / 3)
183|         coeff = max(distance_moyenne / nor_eve , 1) #a ameliorer ##
choix important ##
184|         vec_eve = (1 / nor_eve) * coeff * vec_eve
185|         return vec_eve
186|
187|     def vecteur_evolution(N, individu, evenement, sortie_proche,
attenuation):
188|         vec_eve = vecteur_evenement(individu, evenement, N)
189|         vec_sor = vecteur_sortie(individu, sortie_proche)
190|         vec_evo = vec_sor + attenuation * vec_eve
191|         nor = norme(vec_evo)
192|         return 1 / nor * vec_evo
193|
194|
195|     def correction_debordement(xn, yn, N):
196|         if xn < -N:  xn = -N
197|         if xn > N:   xn = N
198|         if yn < -N:  yn = -N
199|         if yn > N:   yn = N
200|         return np.array([xn, yn])
201|
202|
203|     def nouv_collisions_cree_par_i(foule, i, pos_ancienne,
pos_nouvelle):
```

```
204|     #bousculades crees par i
205|     bousc_max = 9 ## choix important 8 voisins géométriques  ##
206|     ii, pbi = i + 1, 0 #pbi pour personnes bousculées par i
207|     while ii < len(foule) and pbi < bousc_max:
208|         pos = foule[ii]
209|         d_ancienne = distance(pos_ancienne, pos)
210|         d_nouvelle = distance(pos_nouvelle, pos)
211|         if d_nouvelle < 1 and d_ancienne > 1: # 1 ou pas
212|             pbi += 1
213|         ii +=1
214|     return pbi
215|
216| def mise_a_jour_encombrement_sorties(N, foule, ef, sorties,
taille_porte):
217|     nb_sor = len(sorties)
218|     nb_ind = len(foule)
219|     es = [0] * nb_sor
220|     rayon_bloq = taille_porte
221|     for i in range(nb_ind):
222|         if ef[i] == 1:
223|             individu = foule[i]
224|             ind_s = sortie_proche(N, individu, sorties)
225|             d = distance(individu, sorties[ind_s])
226|             if d < rayon_bloq: # rayon_seuil; fameux seuil de
blocage
227|                 es[ind_s] += 1
228|     return es
229|
230| def evolution(N, foule, ef, sorties, es, taille_porte,
evenement, attenuation, densite_bloq):
231|     """mise à jour de ef, es """
232|     """calcul du nombre des nouvelles bousculades/colisions"""
233|     nb_ind, nb_sor = len(foule), len(sorties)
234|     nouvelles_bousculades = 0
235|     nb_ind_sortis = [0] * nb_sor
236|     rayon_bloq = taille_porte
237|     nb_bloq = 1/2 * 3.14 * rayon_bloq ** 2 * densite_bloq
238|     #nb_ind_sortis[j] nombre de personnes sorties par la porte j
239|     for i in [i for i in range(nb_ind) if ef[i] == 1 ]:
240|         individu = foule[i]
241|         pos_ancienne = foule[i]
242|         ind_s = sortie_valide(N, nb_ind, individu, sorties, es,
taille_porte, densite_bloq)
243|         if ind_s == -1:
244|             vec_evo = np.array([0,0])
245|         else:
246|             sortie_proche = sorties[ind_s]
247|             vec_evo = vecteur_evolution(N, individu, evenement,
sortie_proche, attenuation)
248|             #vec_eve = vecteur_evenement(individu, evenement, N)
249|             #vec_sor = vecteur_sortie(individu, ps)
250|
251|         pos_nouvelle = pos_ancienne + vec_evo
252|         xn, yn = pos_nouvelle[0], pos_nouvelle[1]
253|         """calcul du nombre des nouvelles bousculades """
```

```
254|         pbi = nouv_collisions_cree_par_i(foule, i, pos_ancienne,
pos_nouvelle)
255|         # if pbi > 8 : vec_evo = [0,0]
256|         # ou  vec_evo = 1 / ( 1 + sqrt(pbi) ) * vec_evo ## choix
important ##
257|         #puis pos_nouvelle = pos_ancienne + vec_evo
258|         pos_nouvelle = correction_debordement(xn, yn, N)
259|
260|         #Attention à faire avant mise à jour foule
261|         nouvelles_bousculades += pbi
262|         """mise à jour ef en direct plus rapide """
263|         #if ind_s != -1:
264|         d_nou = distance(pos_nouvelle, sorties[ind_s])
265|
266|         dps = taille_porte / 4 # à déterminer ## choix important
##
267|             #dps distance pour sortir
268|         if (d_nou < dps) and nb_ind_sortis[ind_s] < taille_porte
and es[ind_s] < nb_bloq:
269|             #nb_ind_sortis[j] nombre d'individus sortis par la
porte j
270|             ef[i] = 2
271|             nb_ind_sortis[ind_s] += 1
272|                 #es[ind_s] -= 1 # ou pas s'il bouge en même
temps
273|         """Mise à jour de foule """
274|         foule[i] = pos_nouvelle
275|     #print(nb_ind_sortis)
276|     nb_nouv_dehors = sum(nb_ind_sortis)
277|     return nouvelles_bousculades, nb_nouv_dehors
278|
279| def sorties_ouvertes(es, nb_bloq):
280|     so = True
281|     for s in es:
282|         if s < nb_bloq:
283|             return so
284|     return not so
285|
286| def mouvements_foule(N, densite, nb_it_max, nb_sorties,
taille_porte, danger):
287|     foule = creation_foule(N, densite)
288|     sorties = creation_sorties(N, nb_sorties)
289|     evenement = creation_evenement(N)
290|     nb_ind = len(foule)
291|     rayon_bloq = taille_porte  ## choix important ##
292|     densite_bloq = 0.8 ## choix important ##
293|     nb_bloq = 1/2 * 3.14 * rayon_bloq ** 2 * densite_bloq
294|     ef, nb_imm = initialisation_etat_foule(N, foule, evenement,
danger)
295|     es = mise_a_jour_encombrement_sorties(N, foule, ef, sorties,
taille_porte)
296|     nb_bousculades = 0
297|     nb_it = 0
298|     nb_deh = 0
299|     print("nb_ind = ",nb_ind, "nb_imm = ",nb_imm)
```

```
300|     while nb_it < nb_it_max  and nb_deh < nb_ind - nb_imm  and
sorties_ouvertes(es, nb_bloq):
301|         attenuation = 0.5 * danger / (nb_it + 1) #à améliorer
302|         """la répulsion de evenement decroit avec les
itérations, bof...
303|             prendre danger en compte """
304|         if nb_it > 1 and nb_it < 35 and nb_it % 3 == 0:
305|             plt.figure(nb_it) # ne plus faire les visulations si
nb_it_max > 20
306|             visualisation(N, foule, ef, evenement, sorties, es,
taille_porte, densite_bloq)
307|             plt.show(block = True)
308|         print(nb_it, nb_bousculades, nb_deh, es)
309|         nouvelles_bousculades, nb_nouv_dehors = \
310|         evolution(N, foule, ef, sorties, es, taille_porte,
evenement, attenuation, densite_bloq)
311|         nb_bousculades += nouvelles_bousculades
312|         nb_deh += nb_nouv_dehors
313|         es = mise_a_jour_encombrement_sorties(N, foule, ef,
sorties, taille_porte)
314|         nb_it += 1
315|     print(nb_bousculades, nb_deh, sorties_ouvertes(es, nb_bloq))
316|     #visualisation(N, foule, ef, evenement, sorties, es,
taille_porte, densite_bloq)
317|     return nb_bousculades, nb_deh, sorties_ouvertes(es, nb_bloq)
318|
319|
320| mouvements_foule(N = 20, densite = 0.19, nb_it_max = 35,
nb_sorties = 4, taille_porte = 4, danger = 2)
321|
322| def ratio_sortis_fn_densite(N):
323|     X_densite = [d /100 for d in range(10,39,10)]
324|     Y_ratio = []
325|     for d in X_densite:
326|         _, nb_deh, _ = mouvements_foule(N, d, 30,  4 , 4 , 2)
327|         nb_ind = ( 2 * N + 1) ** 2
328|         ratio = nb_deh / nb_ind
329|         Y_ratio.append(ratio)
330|     return Y_ratio
331|
332| #print(ratio_sortis_fn_densite(N = 20))
333|
334|
##########################################################################
#####################
335| def visualisation_trajectoire(depart, sorties, evenement,
nb_iter, N):
336|     """avec débordements possibles """
337|     depart.append(0)
338|     trajectoire = [depart]
339|     for j in range(nb_iter):
340|         individu = trajectoire[-1]
341|         x, y = individu[0], individu[1]
342|         pos = np.array([x, y])
343|         vec_evo = np.zeros(2)
```

```
344|         vec_sor = vecteur_sortie(pos, sorties)
345|         vec_eve = vecteur_evenement(pos, evenement,N)
346|         attenuation = 1 / (j+1)
347|         vec_evo = vec_sor + attenuation * vec_eve
348|         nor_evo = norme(vec_evo)
349|         if nor_evo < 0.00001:
350|             vec_evo = np.zeros(2)
351|             print("vecteur nul")
352|         else:
353|             vec_evo *= ( 1 / nor_evo)
354|         pos = pos + vec_evo
355|         pos = list(pos)
356|         pos.append(0)
357|         trajectoire.append(pos)
358|     visualisation(trajectoire, evenement, sorties, N)
359|     #return evol_ind
360|
361| # ss = [[-5,10],[5,-10]]
362| # ee = [-5,3]
363| # for j in range(10):
364| #     visualisation_trajectoire([-10 + 2 * j,0,0], ss, ee,10,10)
365|
366|
367|
```