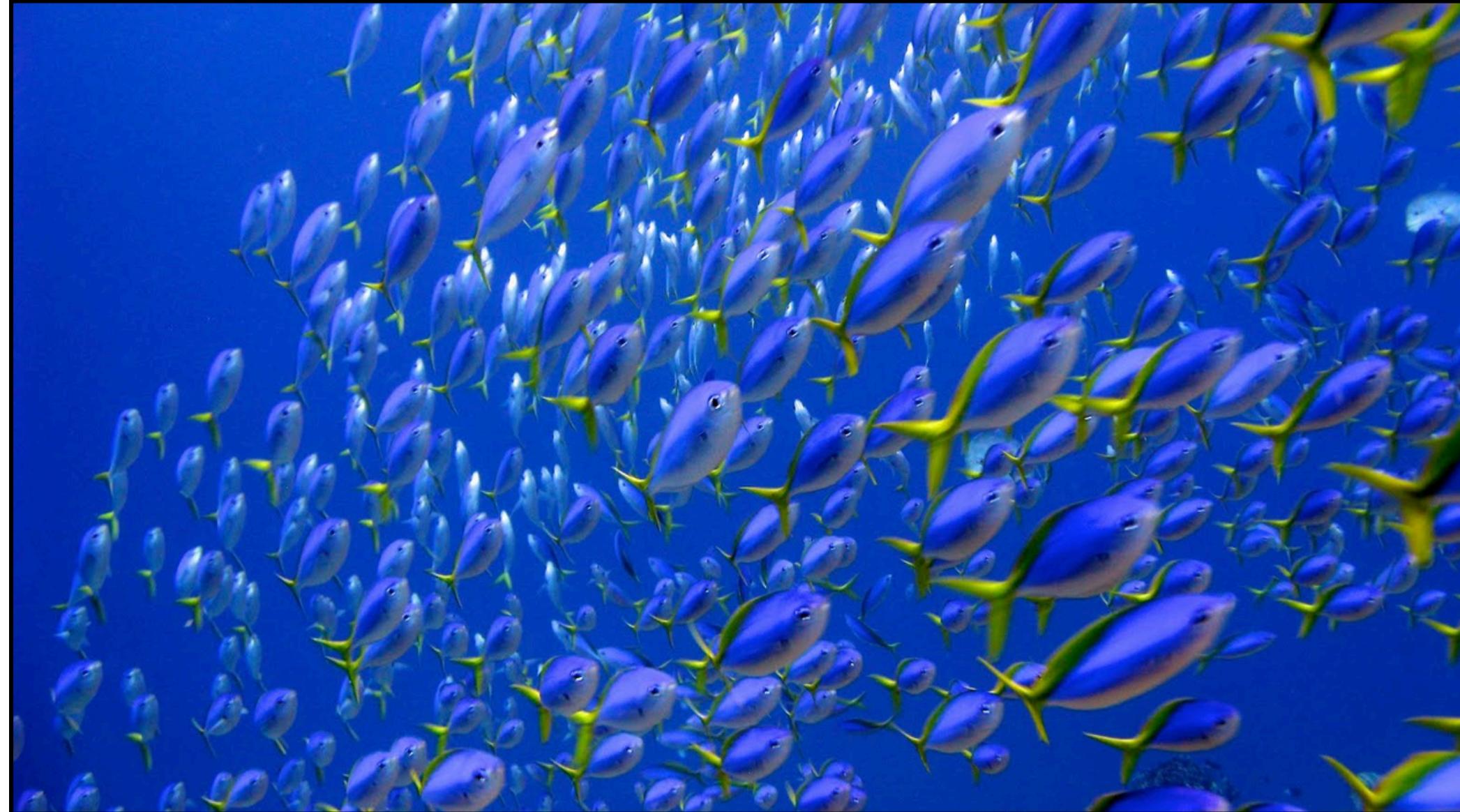


# Modélisation d'un banc de poissons

## Projet 2024-2025



Assala ASSELLALOU

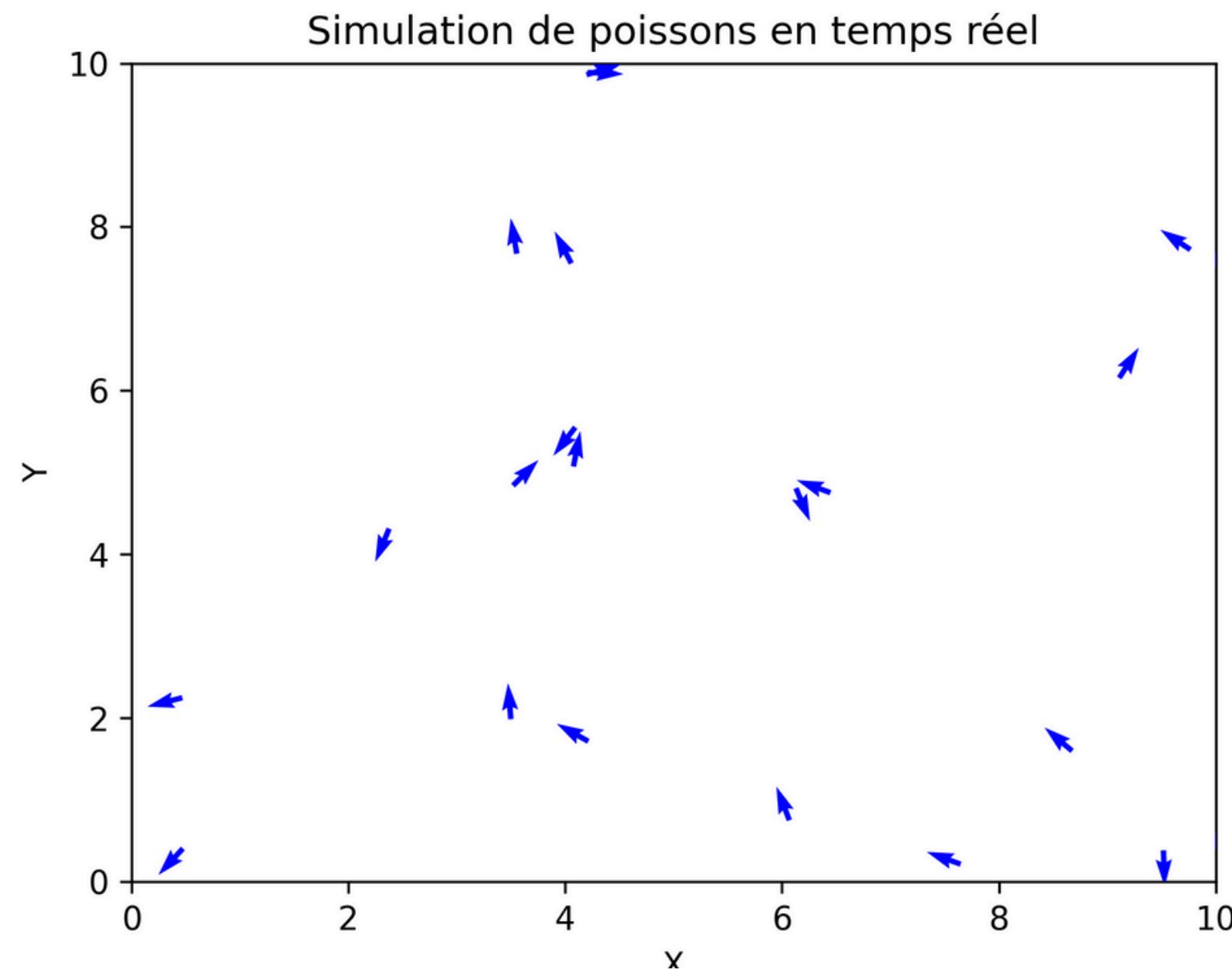
Hedi ZAHAF

# Partie 1 : Mouvement Aléatoire

- Classe Poisson : fichier poisson.py
- Simulation Partie 1 : fichier Partiel.py
- Un poisson est identifié dans cette partie par sa position et sa vitesse
- On utilise l'équation de mouvement :  
**position(t + dt) = position (t) + vitesse (t) \* dt**
- Dans classe Poisson :
  - On initialise les coordonnées de position x et y et les composantes de vitesse Vx et Vy :  
**\_\_init\_\_(self, x, y, Vx, Vy, color='blue')**
  - On écrit une méthode pour déplacer les poissons dans une durée Dt :  
**deplacer(self, Dt)**
  - On écrit une méthode pour gérer les rebonds sur les bords :  
**verifier\_bords(self, xmin, xmax, ymin, ymax)**
  - On écrit une méthode statique pour créer un banc de poissons :  
**creer\_banc(nb\_poissons, xmin, xmax, ymin, ymax, Vmin=-1, Vmax=1)**

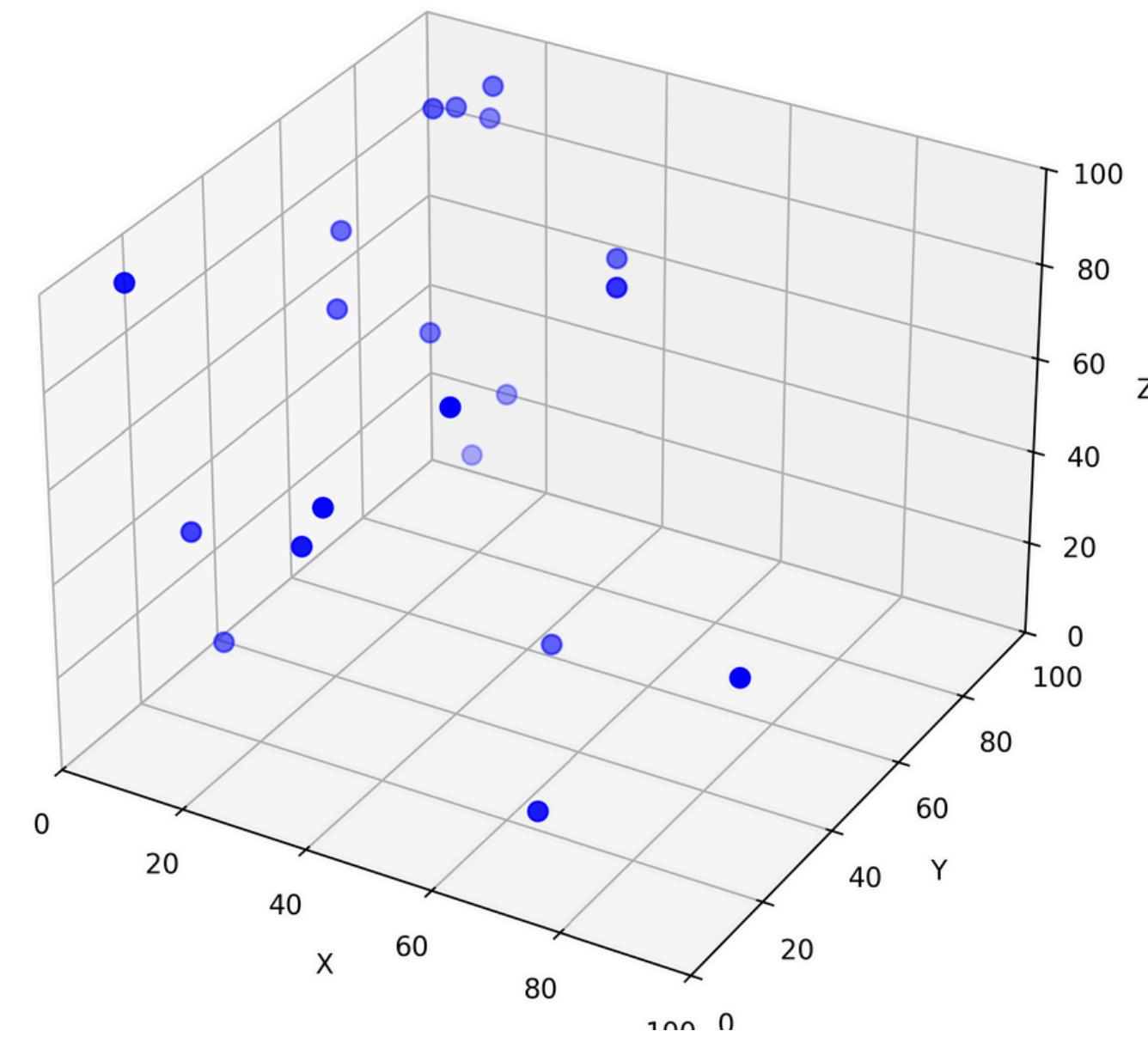
# Partie 1 : Mouvement Aléatoire

Simulation Partie 1 (2D)



Simulation Partie 1 (3D)

Mouvement aléatoire de poissons en 3D

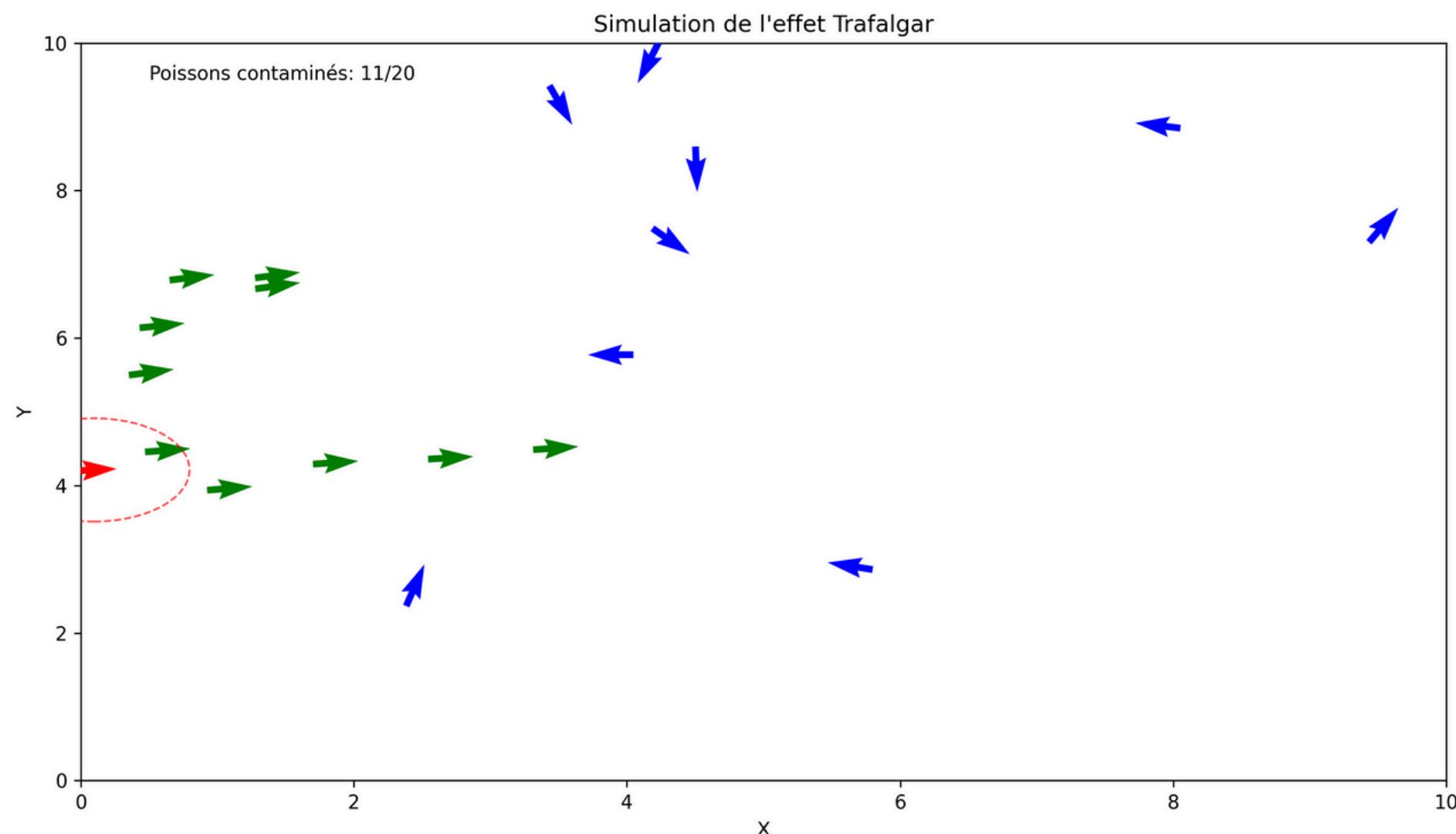


## Partie 2 : Propagation des Mouvements- L'effet Trafalgar

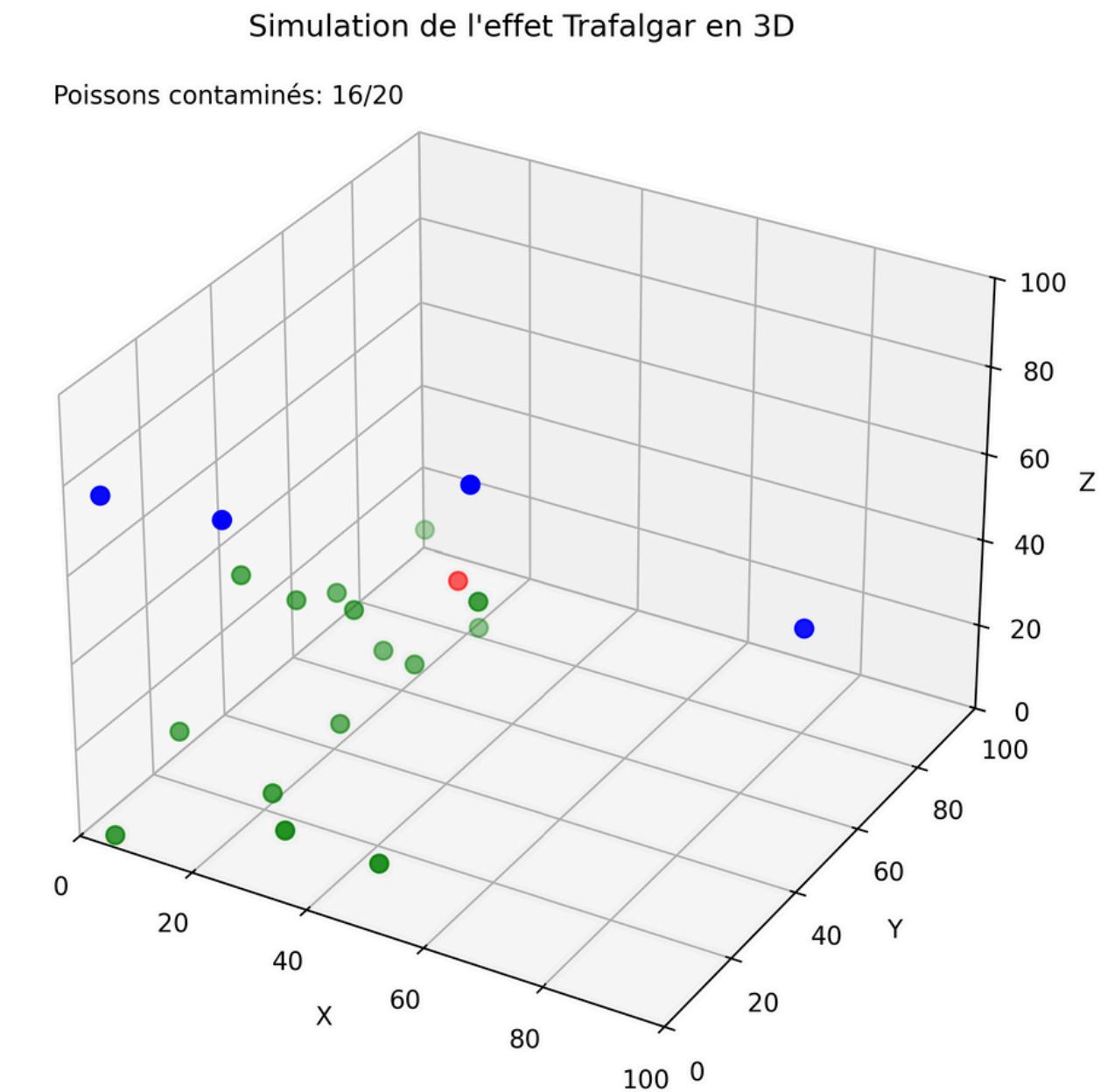
- Classe Poisson : fichier poisson.py
- Simulation Partie 2 : fichier Partie2.py
- Un poisson est identifié dans cette partie par sa position et sa vitesse , mais aussi par son état (contamination , couleur)
- **Comment les poissons réagissent aux mouvements de leurs voisins les plus proches ?**
- **Algorithme :**
  - **1- Sélection du leader** : sélection aléatoire , leader contamine, couleur rouge.
  - **2- Calcul des distances** : afin d'identifier les voisins les plus proches.
  - **3- Propagation du comportement** : Si voisin proche du leader ou poisson contaminé => contamination, changement de vitesse, changement de couleur.
- **Etape 1 : leader = choix\_aleatoire(particules)**
- **Etape 2 : distance(pi, pj) = np.sqrt((xi - xj)^2 +(yi -yj)^2)**
- **Etape 3 : \* si distance(p,c) < limite => p.vitesse += variation\_aleatoire & p.couleur = 'vert'**  
**\* en plus de la mise à jour position et vérification de frontière**

# Partie 2 : Propagation des Mouvements- L'effet Trafalgar

Simulation Partie 2 (2D)



Simulation Partie 2 (3D)

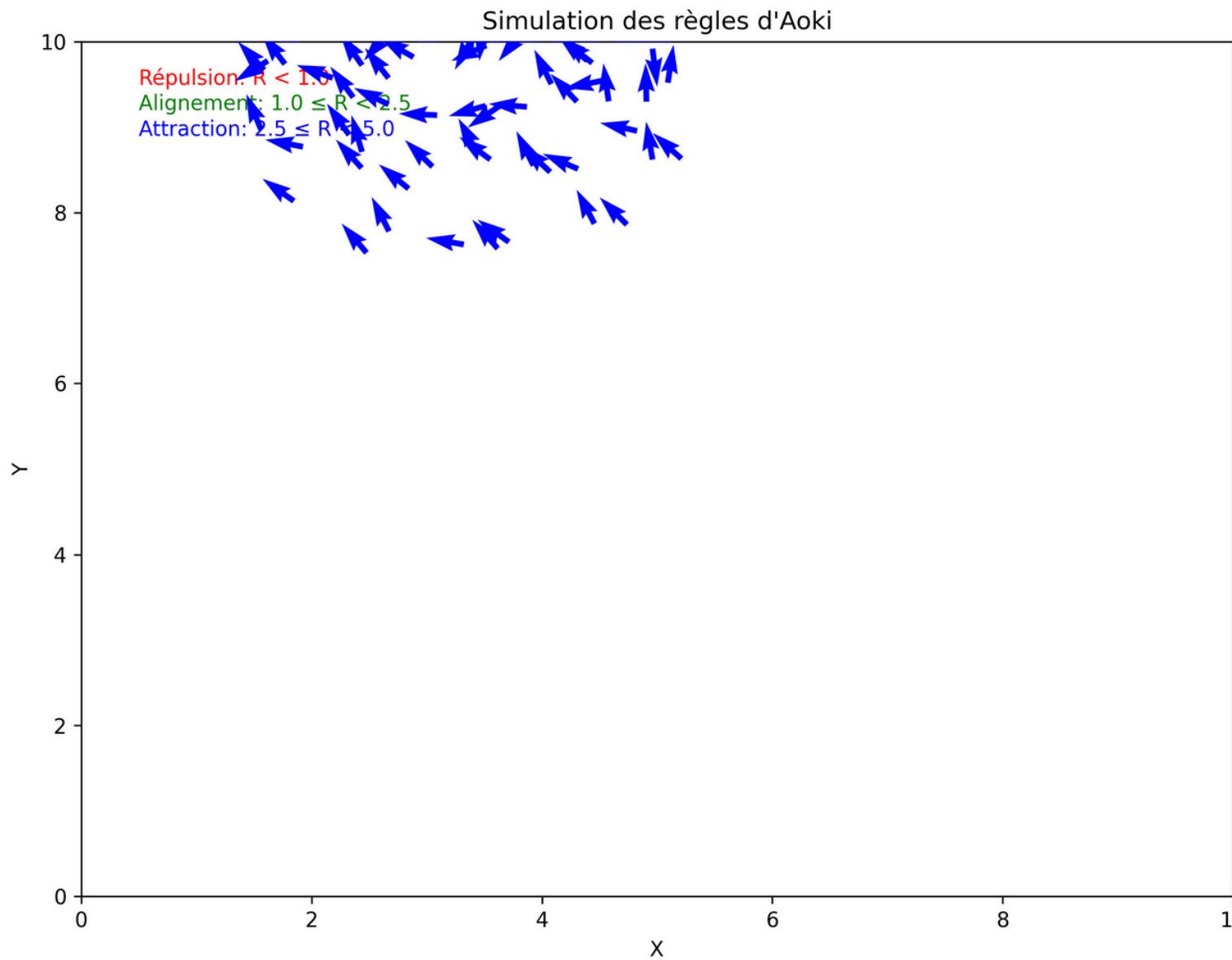


# Partie 3 : Règles Comportementales de Aoki

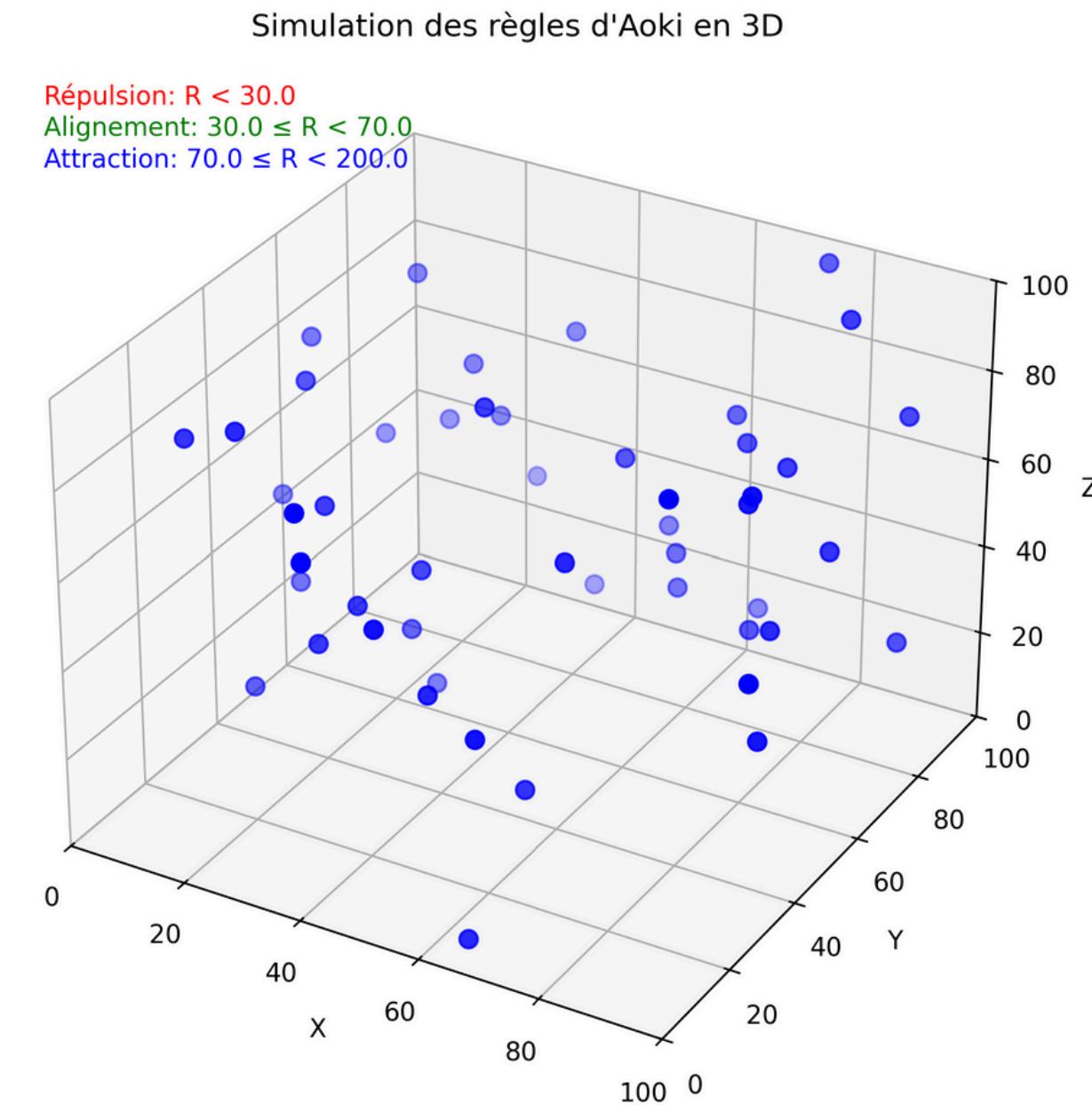
- Classe Poisson : fichier poisson.py
- Simulation Partie 3 : fichier Partie3.py
- Un poisson est identifié dans cette partie par sa position et sa vitesse
- **Explication des mouvements coordonnées des bancs de poissons, basé sur trois règles comportementales principales :**
  - 1- **Evitemen / Répulsion** : les poissons évitent les autres poissons très proches
  - 2- **Alignement** : les poissons s'alignent avec ceux qui ont une distance intermédiaire
  - 3- **Attraction** : les poissons se rapprochent des autres poissons éloignés
- **Etapes :**
  - 1- **Calcul des distances et identifications des voisins**
  - 2- **Applications des forces**
  - 3 - **Mise à jour des vitesses**
- **Formules : (contraintes sur la distance suivant les rayons)**
  - **F\_repulsion = - k\_repulsion \* vecteur\_d / norme\_d**
  - **F\_alignement = 1 / N\_alignement \* somme(vitesses) #aligner la vitesse avec celle des voisins**
  - **F\_attraction = k\_attraction \* vecteur\_d / norme\_d**

# Partie 3 : Règles Comportementales de Aoki

Simulation Partie 2 (2D)

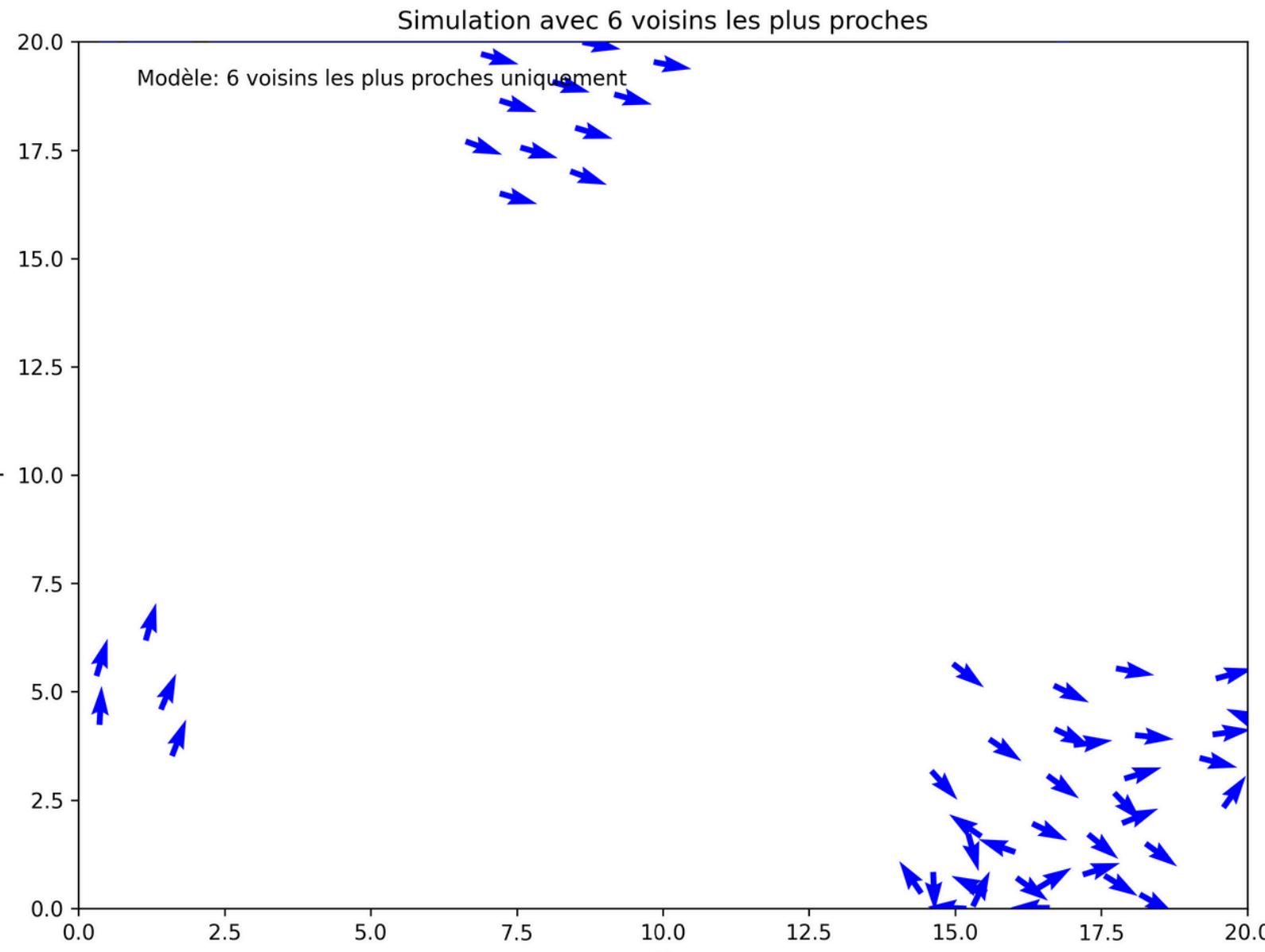


Simulation Partie 2 (3D)

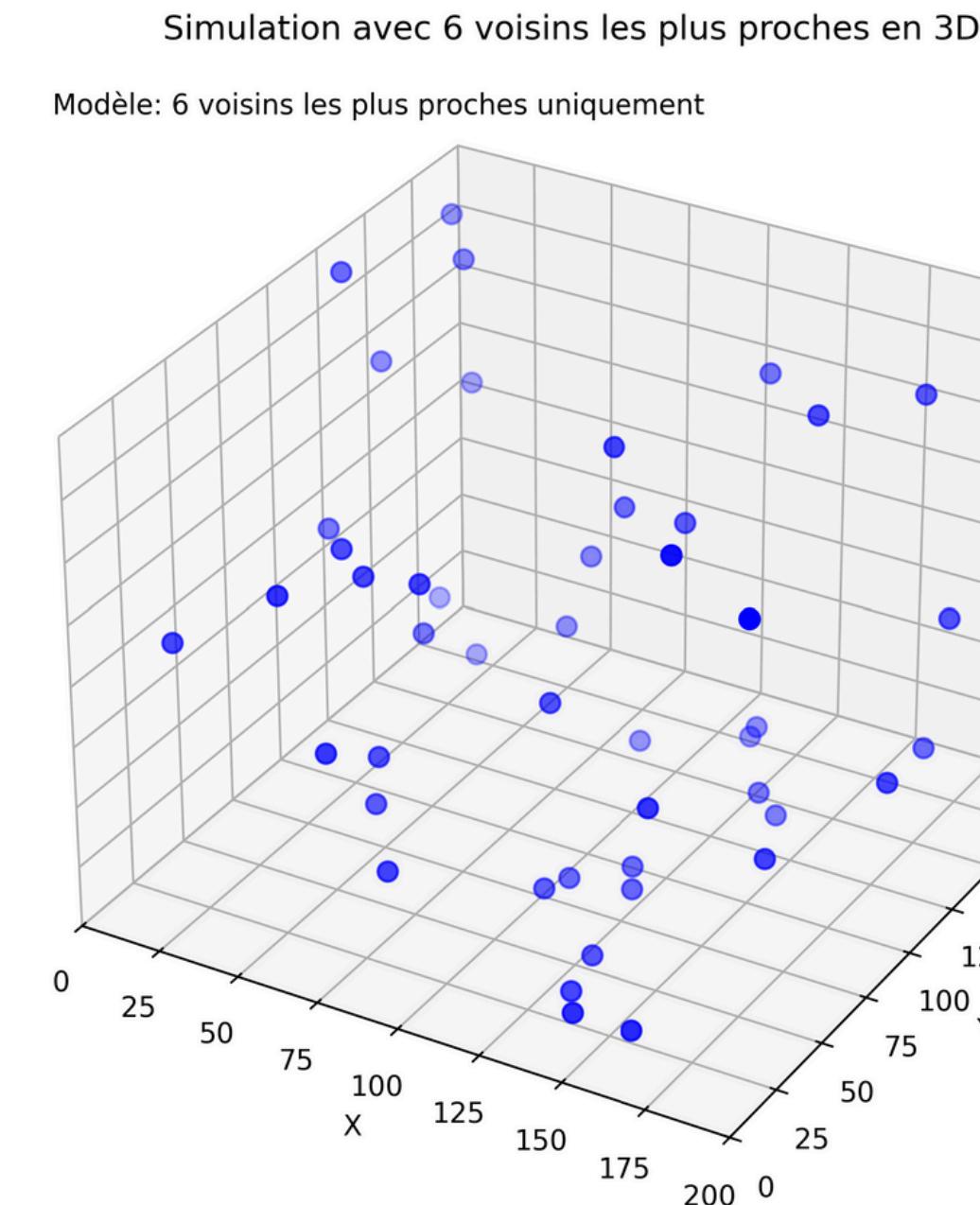


# Partie 4 : Influence de la Densité

Simulation Partie 4 (2D)

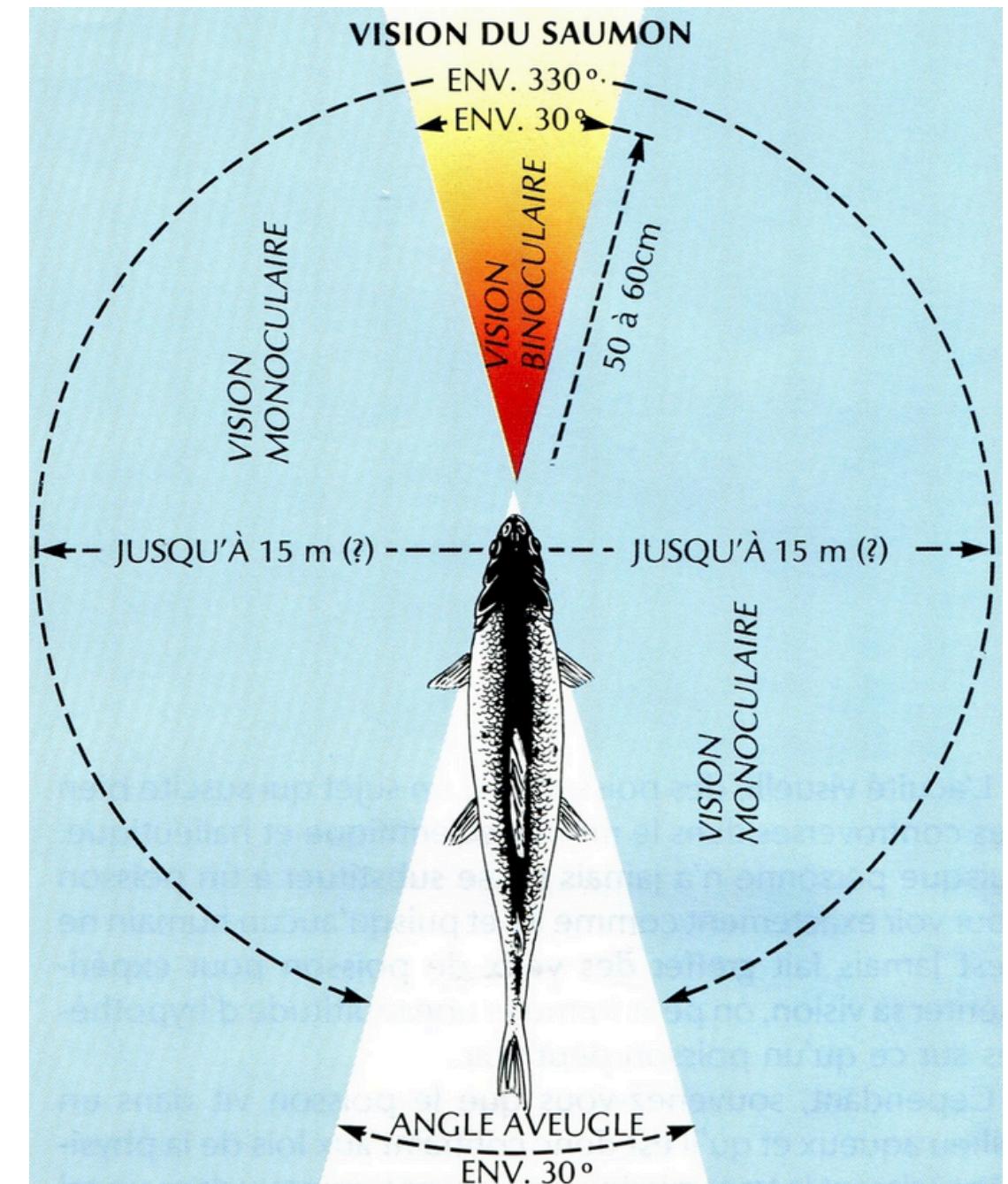


Simulation Partie 4 (3D)



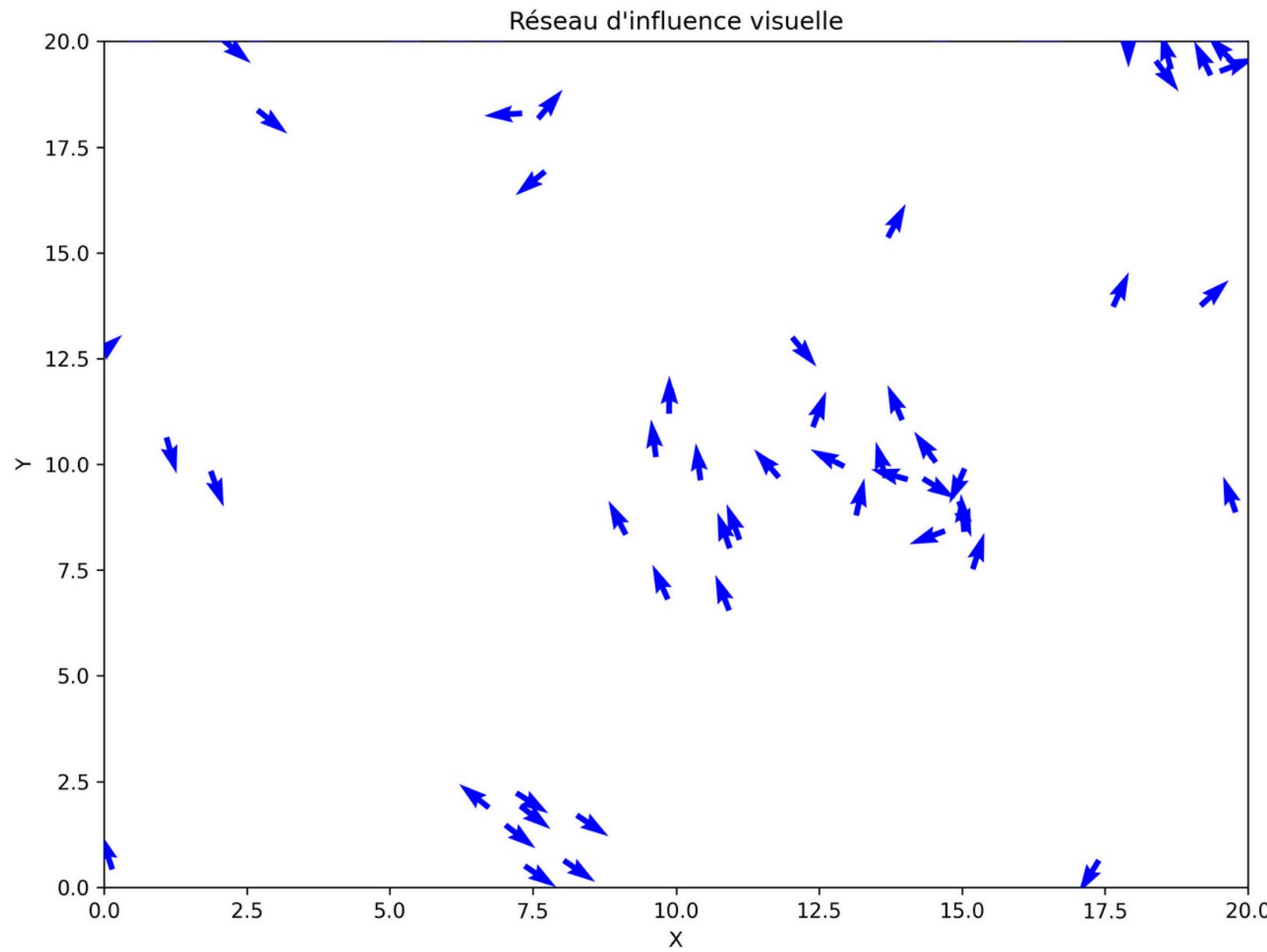
# Partie 5 : Réseau d'Influence

- Connexions visuelles
- Etapes :
  - 1- Détection particules visibles
  - 2- Lancer de rayon dans cone de vision
  - 3- Application des forces comportementales
  - 4- Mise à jour de la vitesse

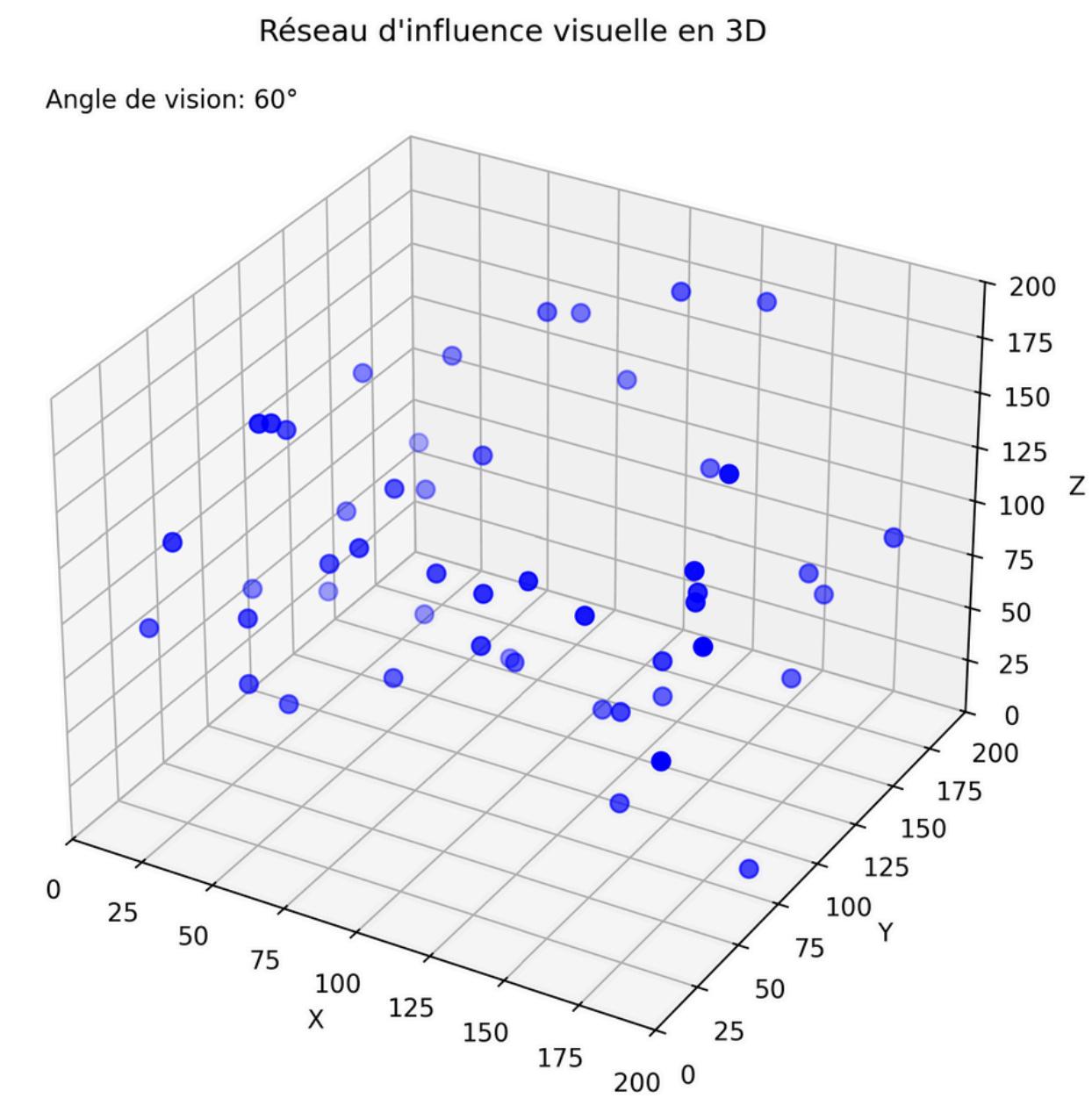


# Partie 5 : Réseau d'Influence

Simulation Partie 5 (2D)



Simulation Partie 5 (3D)



# Annexe

- **Python** : Langage principal pour la simulation et l'animation
- **Classes personnalisées (Poisson, Poisson3D)** : Comportements individuels et collectifs
- **NumPy** : Calculs vectoriels (positions, vitesses) et manipulation de matrices
- **Matplotlib** : Visualisation et animation 2D/3D des bancs de poissons
- **Random** : Initialisation aléatoire des états initiaux (position, vitesse)
- **SciPy KDTree** :
  - SciPy KDTree
  - Structure optimisée pour recherche de voisins en 2D/3D
  - Accélère la détection des poissons proches (vs. double boucle  $O(N^2)$ )
  - Recherche en  $O(\log N)$  par requête grâce à `query_ball_point`
  - Indispensable pour simuler les interactions locales (répulsion, alignement, attraction)