



UNIVERSITÉ  
TOULOUSE III  
PAUL SABATIER  
Université  
de Toulouse



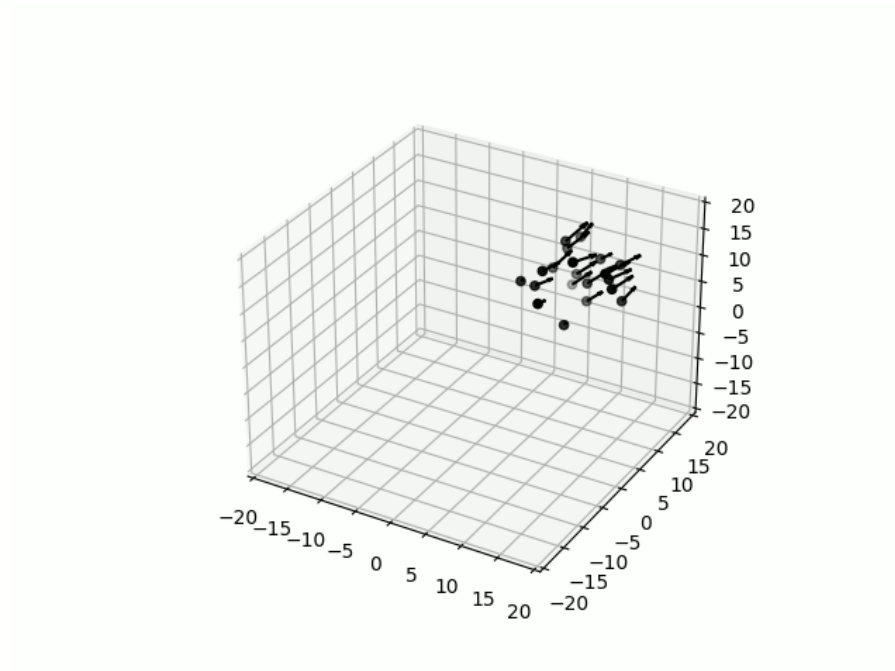
LICENCE 3 DE PHYSIQUE, CHIMIE, ASTROPHYSIQUE, MÉTÉOROLOGIE ET  
ÉNERGIE

---

RAPPORT DU PROJET NUMÉRIQUE

# MODÈLE DE VICSEK

Alexis PEYROUTET, Antoine ROYER



Janvier – Juin 2023

---

# Table des matières

<b>Introduction</b>	<b>2</b>
<b>1 Présentation et explications</b>	<b>3</b>
<b>2 Méthode employée</b>	<b>5</b>
2.1 Classes et méthodes . . . . .	5
2.2 Création et manipulations d'agents . . . . .	5
2.3 Création et manipulations de groupes . . . . .	6
<b>3 Premières interprétations physiques</b>	<b>7</b>
3.1 Animations en fichier GIF . . . . .	7
3.2 Mouvements de groupe . . . . .	8
3.3 Paramètres en jeu . . . . .	9
3.3.1 Cône de vision . . . . .	9
3.3.2 Paramètre de bruit . . . . .	10
<b>4 Résultats expérimentaux</b>	<b>12</b>
4.1 Résultats historiques de VICSEK . . . . .	12
4.1.1 Paramètre d'alignement en fonction du bruit . . . . .	12
4.1.2 Paramètre d'alignement en fonction de la densité . . . . .	14
4.2 Au-delà du modèle de VICSEK . . . . .	15
4.2.1 Création d'un agent leader . . . . .	15
4.2.2 Mise en place d'une prédation et du paramètre de sensibilité . . . . .	16
4.2.3 Système évolutif . . . . .	17
4.2.4 Ajout d'obstacles . . . . .	17
<b>Conclusion</b>	<b>19</b>
<b>A Classes et fonctions de générations</b>	<b>21</b>
<b>B Code d'utilisation</b>	<b>41</b>

# Introduction

Notre sujet est intitulé « Modèle de VICSEK ». Le but de ce projet numérique est de reproduire de manière numérique le modèle de VICSEK.

Le modèle de VICSEK [1] a été créé par le scientifique Tamás VICSEK [2]. Il s'agit d'un physicien hongrois connu pour ses contributions à la physique statistique, à la biologie et à la dynamique des systèmes. Il est né le 10 mai 1948 (74 ans aujourd'hui) à Budapest. Il est aujourd'hui professeur à l'Université Eötvös LORÁND de Budapest. Ce brillant physicien est d'ailleurs un des membres de l'Académie hongroise des sciences et a reçu de nombreux prix pour ses contributions à la physique, notamment le prix SZÉCHENYI (en 1999) ou encore le prix Lars ONSAGER (en 2020).

Mais Tamás VICSEK est surtout connu pour son travail sur les systèmes auto-organisés ; ces modèles permettent d'étudier les mouvements collectifs. Il a ainsi travaillé sur le comportement d'agents individuels qui interagissent avec d'autres agents aux alentours, ses observations montrent que des motifs de mouvements collectifs émergent d'eux-même.

Le groupe se déplace alors de manière coordonnée sans qu'il n'y ait de leader comme on peut l'observer notamment dans la migration des grues. Nous pouvons citer comme exemples : les bancs de poissons, les regroupements de certains oiseaux, les essaims d'insectes, ou encore le mouvement de foules.

Le premier modèle de VICSEK voit ainsi le jour en 1995.

# Chapitre 1

## Présentation et explications

Le modèle de VICSEK permet d'étudier un groupe d'agents qui se déplace dans un espace.

Chacun des agents a une vitesse donnée (en norme et en direction) et va interagir avec ses voisins. Ce qui concrètement se traduit par des changements concernant la norme et la direction de la vitesse.

Nous nous attendons alors à observer la création d'un mouvement de groupe du aux interactions entre les agents.

Cependant, les agents observables dans la vie réelle ne suivent pas toujours le groupe à la perfection, et il peut arriver qu'un agent s'écarte, de manière aléatoire, des autres. C'est pour cela que VICSEK a introduit une notion de bruit dans son modèle. En effet, à chaque pas de temps, chaque agent va prendre la direction moyenne des agents autour de lui et à cette direction va venir se superposer un bruit qui le fera peut-être dévier dans une autre direction.

En augmentant significativement le bruit, le groupe perd son mouvement collectif et les agents prennent alors des directions aléatoires.

Le modèle de VICSEK s'implémente très simplement puisqu'il se réduit à deux équations [3] :

$$\begin{cases} \Theta_i(t + dt) &= \Theta_{j|r_i-r_j|<r} + \eta_i(t) \\ r_i(t + dt) &= r_i(t) + v_i \Delta t \end{cases}$$

dans lesquelles  $r_i$  la position de chaque individu donnée par un vecteur de position, nous prendrons  $i$  comme indice de l'agent en question et  $t$  le temps. Nous noterons également  $\eta$  le bruit et  $\Theta$  pour l'angle définissant la direction de sa vitesse. Ici,  $\Theta_{j|r_i-r_j|<r}$  nous indiquera la direction moyenne des vitesses des agents dans un cercle de rayon  $r$ . L'indice  $j$  représentera alors l'ensemble des voisins de  $i$  compris dans ce cercle.

Ce qui est intéressant, c'est que nous pouvons, en modifiant certains paramètres du système étudié, observer un mouvement de foule plus fort ou plus faible. Nous pourrons alors jouer sur la surface et les dimensions du plan étudié, le nombre d'agent et donc par conséquent la densité de population et même le bruit.

Le modèle de VICSEK est important pour étudier le comportement de certains animaux en biologie ou encore l'étude des foules. Ce modèle peut même être utile à la construction de bâtiment. En effet, le comportement des foules peut être intéressant dans la conception d'entrées et de sorties d'un espace fermé, notamment dans un moment de panique. La foule va s'éloigner du danger est emprunter les sorties. Il est alors crucial de prévoir le comportement des agents pour placer les sorties de manière à ce que le débit d'agent sortant soit le plus important possible.

Nous pouvons également retrouver le modèle de VICSEK dans la robotique. C'est un précieux outil pour la technologie du monde moderne. Il peut être utilisé dans des programmes informatiques qui gèrent le déplacement de systèmes de robots (comme les drones).

C'est avec tout cela que nous avons essayé, à travers ce projet, de reproduire numériquement des mouvements collectifs et ainsi étudier le modèle de VICSEK.

# Chapitre 2

## Méthode employée

### 2.1 Classes et méthodes

Pour ce projet, la programmation orientée objet s'est naturellement imposée. Nous utilisons ainsi deux classes appelées **Agent** et **Group** qui fixent respectivement les paramètres de l'agent et du groupe. Assez simplement, la classe **Group** contient une liste d'instances de la classe **Agent** et permet de les représenter dans l'espace et le temps. La classe **Agent** permet d'encapsuler toutes les données de chaque agent : position, vitesse et direction, bruit, champ de vision etc.

Ainsi, nous pouvons retrouver dans chaque classe, plusieurs méthodes qui vont nous aider à mieux définir les groupes et les agents ainsi qu'à les faire évoluer.

### 2.2 Création et manipulations d'agents

Pour représenter nos agents, nous avons créé une classe qui contient la position de l'agent, sa direction, ainsi que sa vitesse.

À ces paramètres de bases, nous avons rajouté la distance de vision qui permet à chaque agent d'avoir une vue limitée d'une part, mais aussi variable d'un agent à l'autre, ce qui nous permet de simuler une population avec des aveugles par exemple ou des individus d'âges différents (ce qui n'est pas le cas dans le modèle de VICSEK) . Les agents ont également un champ de vision unique. Cela caractérise leur angle de vue.

Le bruit d'un agent caractérise l'écart angulaire aléatoire entre sa direction et la direction moyenne de ses voisins. Un bruit fort entraîne donc des écarts importants et les groupes se dissocient plus facilement (nous parlerons du bruit plus en détail dans la suite du rapport).

Avec l'apparition des agents répulsifs (voir chapitre 4), nous avons également rajouté une sensibilité à ces agents, ce qui rend compte d'une peur des agents répulsifs. Ainsi un agent normal avec une peur nulle ne fuira pas les agents répulsifs. Nous avons ainsi étudié des groupes avec des valeurs limites de bruits et de sensibilité.

Les agents sont également muni d'un attribut qui caractérise leur type, cela permet de savoir si l'agent est un agent normal, leader, répulsif ou un obstacle. En effet, les obstacles (voir chapitre 4) sont gérés comme des agents répulsifs immobiles et qui ne peuvent pas tuer (contrairement aux agents répulsifs). La répulsion stérique n'ayant pas été prise en compte, les agents peuvent théoriquement traverser un obstacle, nous avons dit forcer les obstacles à avoir la priorité sur les autres agents. Autrement dit, si un agent normal fuit un agent répulsif et qu'il voit un mur, il fera demi tour car le mur est plus puissant que l'agent répulsif. Dans ce genre de cas, il arrive que l'agent normal fasse des aller-retours entre le mur et le prédateur jusqu'à sa mort.

Les agents sont également munis de plusieurs méthodes qui permettent de les manipuler le plus simplement possible.

Nous avons commencé par définir la soustraction comme étant la distance entre les deux agents, ainsi `agent_1 - agent_2` nous renvoie la distance séparant les deux agents ce qui simplifie les écritures dans la suite du programme.

De manière assez standard nous avons créé une méthode `Agent.copy` qui renvoie une copie profonde de l'objet ce qui nous permet de nous affranchir des problèmes d'alias de Python.

La dernière méthode de cette classe est `Agent.next_step`, elle permet de faire évoluer l'agent en fonction de ses voisins qu'il faut donner en argument. C'est dans cette méthode que sont implémentées les équations en qui régissent le modèle.

## 2.3 Création et manipulations de groupes

À l'instar des agents, les groupes sont munis d'un certain nombre d'attributs.

Le plus important d'entre eux est sans doute la liste d'agents. En effet, cette liste contient les instances des agents qu'il faut faire évoluer.

Les groupes ont également une liste d'agents morts et qui correspondent aux agents qui ont été touchés par des agents répulsifs. Les stocker permet de faire des statistiques sur ces agents à la fin de la simulation.

Les deux derniers attributs concernent l'espace : la longueur, la dimension (2 ou 3) et la densité du groupe.

La classe `Group` contient cependant bien plus de méthodes que les agents.

Tout d'abord, nous avons écrit une méthode qui permet de rajouter des agents au groupe. Cela est pratique en particulier pour rajouter des agents spéciaux. Nous pouvons ainsi générer un groupe de 50 agents et rajouter un agent répulsif.

Comme pour la classe `Agent`, nous avons également créé une méthode de copie profonde qui nous permet de dupliquer le groupe en nous affranchissant des problèmes d'alias.

La méthode `Group.get_neighbours` retrouve tous les agents voisins d'un agent donné en argument en fonction de ses caractéristiques.

Pour la représentation graphique, nous avons deux méthodes : `Group.compute_figure` et `Group.compute_animation` qui renvoient respectivement une image et une animation. Mais, de manière à faire des tests sans forcément avoir une trace sous forme d'image ou d'animation, nous avons également fait une méthode `Group.run` qui permet de faire évoluer le groupe.

La dernière méthode a un sens un peu plus physique puisqu'elle correspond au paramètre d'alignement. Ce dernier est défini pour  $N$  agents comme :

$$\sum_{i=1}^N \frac{\mathbf{v}_i}{v_i}$$

où  $\mathbf{v}_i$  est le vecteur vitesse de l'agent  $i$  et où  $v_i$  est la norme ce de même vecteur. Dans son papier VICSEK divise la somme des vitesses par le nombre d'agents multipliés par la norme de la vitesse ce qui présuppose que tous les agents n'ont pas la même vitesse. Dans notre implémentation, nous avons choisi de ne pas imposer une vitesse unique et tous nos agents ont des vitesses qui leur sont propres.

# Chapitre 3

## Premières interprétations physiques

### 3.1 Animations en fichier GIF

Pour pouvoir observer un mouvement, il est plus utile de regarder une vidéo que des images. Nous avons alors créé une méthode : `Group.compute_animation` qui utilise la classe `Artist_animation` du module `matplotlib` [4]. Nous arrivions alors à générer des fichiers GIF avec le nombre de frames souhaité.

Après avoir généré plusieurs animations avec des groupes de tailles différentes, nous avons pu déjà tirer quelques conclusions. En effet, nous observions des mouvements de groupes. Les agents qui avaient des positions de départ aléatoires, sont influencés les uns les autres selon la distance avec leurs voisins.

On observe d'ailleurs des mouvements collectifs plus importants quand la densité d'agent est plus forte. En revanche, quand les agents sont moins nombreux dans un même espace, nous observons davantage de formations de petits groupes ou des agents solitaires. Cela s'explique par le fait que les agents ne se voient plus.

Nous pouvons régler ce paramètre de densité avec `length` qui correspond à la longueur de l'espace considéré.

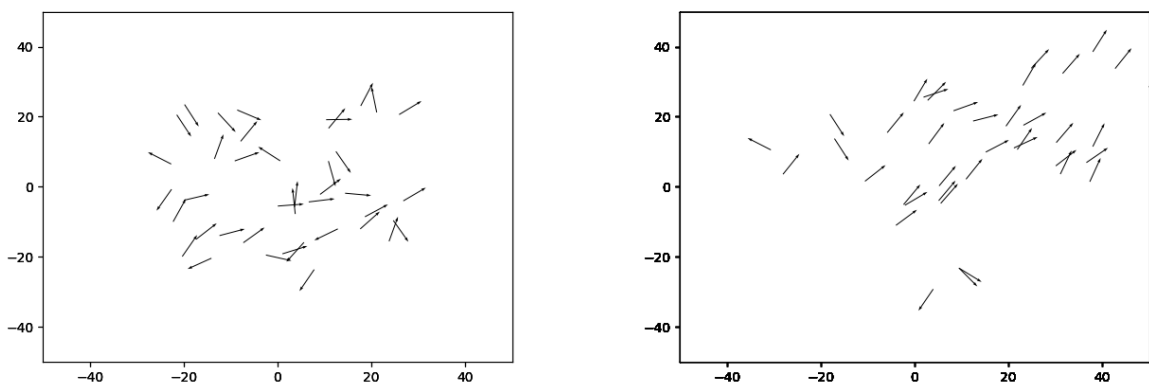


FIGURE 3.1 – Début et fin d'une animation 2D

Nous avons très vite privilégié la représentation 2D pour la suite du projet. En effet, celle-ci permet d'observer plus facilement le comportement des agents, et les différents groupes créés.



## 3.2 Mouvements de groupe

Pour mieux observer les mouvements de groupe, nous avons décidé de mettre une couleur à nos agents selon leur direction. Cela permet de mieux visualiser les différents groupes et de s'affranchir des flèches qui étaient devenues gênantes pour bien discerner le mouvement collectif à haute densité.

Nous avons alors créé une fonction appelée `get_colors`. Avec une boucle `for` et 361 itérations, nous balayons les 360 degrés de l'espace considéré. Avec une suite de `if` et `elif`, nous répartissons la gamme de couleur sur l'ensemble des angles.

Puis, en appelant cette fonction, dans la méthode `get_color` de la classe `Agent`, nous pouvons en fonction de l'angle entre l'horizontale ascendante et le vecteur vitesse de l'agent considéré, associé une couleur particulière.

Nous avons gardé cette représentation de l'angle des agents pour tout le reste du projet.

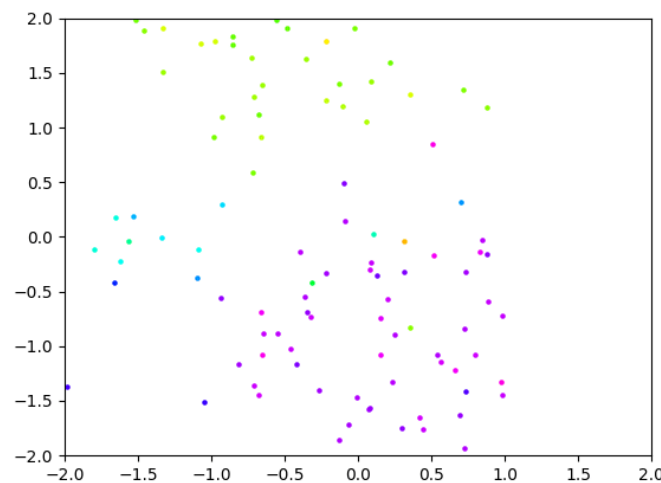


FIGURE 3.2 – Image colorée pour la visualisation de groupe

Sur cette nouvelle figure 3.2, nous voyons bien les différents groupes créés. On distingue encore mieux les mouvements lors de l'animation.

Nous avons d'ailleurs pu observer une organisation intéressante des agents sur certaines animations (voir les 2 figures suivantes). En effet, les agents se regroupent premièrement en plusieurs petits groupes (sur l'image de gauche de la figure 3.4, on discerne en effet trois groupe principaux en violet, vert et cyan). Enfin, les petits groupes s'unissent pour former un seul et même grand groupe.

Ceci montre encore un fois très bien l'influence entre les agents.

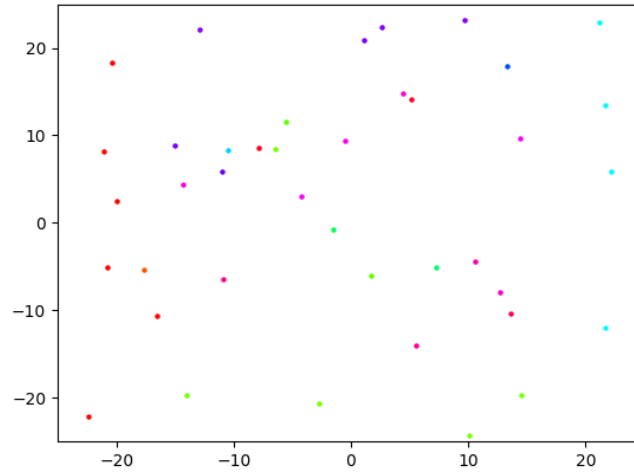


FIGURE 3.3 – Image de départ

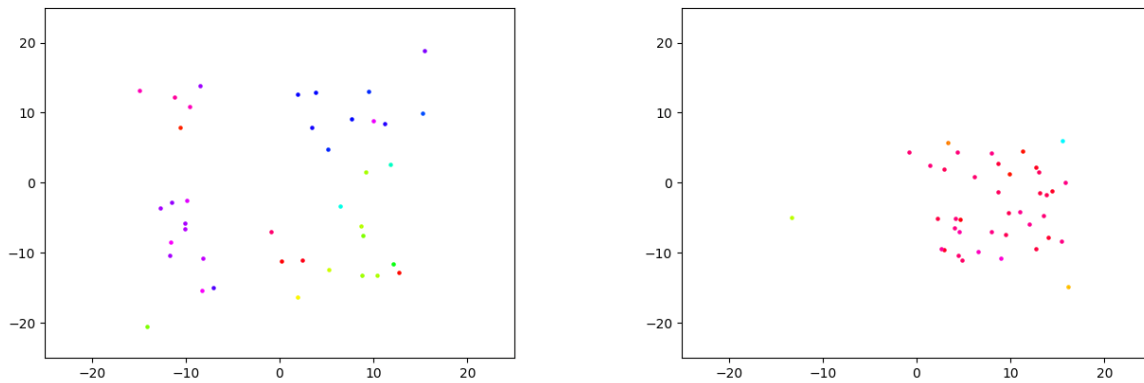


FIGURE 3.4 – Formations de petits groupes puis d'un seul et même groupe

### 3.3 Paramètres en jeu

#### 3.3.1 Cône de vision

Afin de mieux visualiser ce que peut voir un agent en particulier, nous avons alors décidé de rajouter un cône de vision.

Pour ce faire, nous avons utilisé le module `import matplotlib.patches` qui nous a permis de tracer ces cônes en 2 dimensions. Nous nous sommes servis de ce module, qui demande la position et l'angle de vue de l'agent en radian. Nous avons alors converti au préalable l'angle de vue (paramètre `field_sight`) en radians.

Puis, avec le module `matplotlib.collections`, nous avons fait apparaître le cône directement sur la figure générée.

Nous sommes ainsi capable de mieux voir comment un agent réagit selon ce qu'il voit.

Ainsi, un agent est considéré comme voisin s'il est dans le cône de vision de l'agent testé. En refaisant le même test que précédemment, nous observons que les agents restent alignés moins longtemps.

En effet, les agents étant plus sensible à l'orientation pour voir les autres, si le bruit aug-

mente, les agents vont avoir des déviations angulaires plus importante et peuvent donc perdre de vue les autres agents plus facilement ce qui fait chuter le paramètre d'alignement plus rapidement.

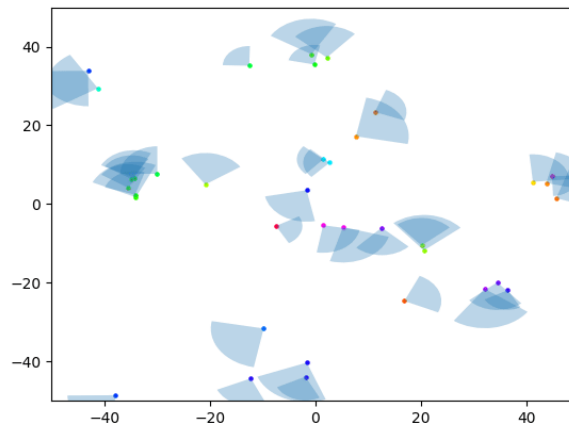


FIGURE 3.5 – Image de l’animation générée avec cône de vision

Nous avons cependant préféré retirer la représentation visuelle de ce cône dans la suite du projet. En effet, l’animation générée devenait trop chargée et riche en informations. Il était alors difficile de bien observer le comportement des agents.

### 3.3.2 Paramètre de bruit

Le bruit correspond à l’influence des voisins sur un agent du groupe. C’est un paramètre essentiel du modèle de VICSEK. Nous le retrouvons (sous la lettre grecque  $\eta$ ) dans les équations qui définissent le modèle dont nous avons parlé dans le chapitre 1.

Plus le bruit est fort, plus celui-ci aura tendance à ne pas s’occuper de ses voisins et prendre sa propre direction. A l’inverse, avec un bruit qui tend vers zéro, l’agent aura tendance à imiter ses voisins et prendre une direction similaire aux agents autour de lui.

Nous avons déjà pu observer l’impact du bruit sur les populations. Nous voyons alors que ce paramètre est capital pour l’observation de mouvements collectifs. Si celui-ci est trop fort, les agents feront route seuls et ne s’occuperont pas du mouvement des voisins. En revanche, les mouvements collectifs seront davantage présents avec un bruit qui tend vers zéro.

Ce paramètre perturbe ainsi la communication entre les agents et affecter leur capacité à se coordonner efficacement. Par exemple, si un agent ne peut pas détecter les mouvements de ses voisins en raison d’un niveau de bruit élevé, il peut suivre une trajectoire différente et rompre la cohérence du groupe.

Pour bien comprendre, nous avons créer cette figure en mettant un bruit très fort :

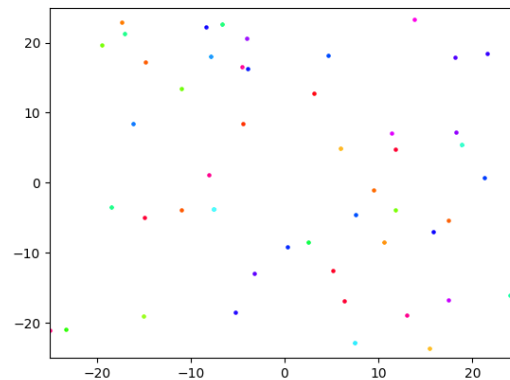


FIGURE 3.6 – Visualisation de l’impact du bruit (bruit fort dans ce cas)

Avec cette image, nous voyons bien ici que les agents ont des directions plutôt différentes. Nous avons un bruit qui est fort dans ce cas, et nous remarquons que les agents ont tendance à prendre des directions sans s’occuper de leurs voisins.

En effet, la cohésion du groupe est significativement réduite. Nous pouvons d’ailleurs remarquer que les agents, de ce fait occupent la quasi totalité de l’espace défini. Au contraire des mouvements de groupe qui font que les agents sont plus regroupés à un endroit précis.

De plus, nous avons également remarqué que certains agents, sous l’influence du bruit changent brutalement de direction au cours de leur déplacement.

Il est alors évident que le bruit est un paramètre essentiel dans le modèle de VICSEK.

# Chapitre 4

## Résultats expérimentaux

### 4.1 Résultats historiques de VICSEK

#### 4.1.1 Paramètre d'alignement en fonction du bruit

Nous avons commencé par chercher à reproduire les résultats obtenus par VICSEK en reprenant les mêmes paramètres.

Chaque agent n'est ainsi influencé que par ses plus proches voisins et évolue dans un espace torique. En laissant la densité constante et en faisant varier le bruit pour voir son influence sur le paramètre d'alignement, nous observons alors un profil de transition de phase.

Nous avons alors créer de nouvelles fonctions qui permettent de calculer ce paramètre d'alignement. La première fonction `op_noise` renvoie un tuple composé du bruit et des valeurs du paramètre d'alignement (pour une densité fixe). Il en va de même pour la seconde fonction `op_density` qui renvoie un tuple composé de la densité et des valeurs de paramètre d'alignement mesuré (pour un bruit fixe).

Bien sûr, pour calculer ce paramètre, nous nous sommes servi de la formule dont nous avons parlé dans la partie 2.3.

Sur la figure suivante chaque points est la moyenne sur cinq mesures de 200 pas et 50 agents.

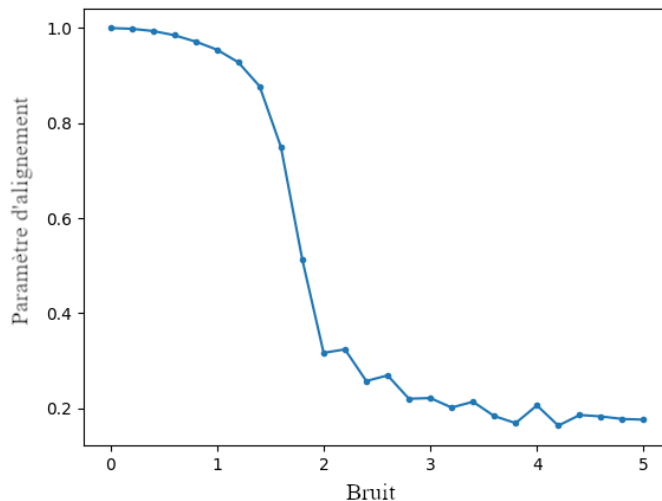


FIGURE 4.1 – Paramètre d'alignement en fonction du bruit

Pour un bruit très faible, les agents sont alignés avec un paramètre d'alignement de 1 ce qui est cohérent puisqu'ils vont s'aligner sans aucune part d'aléatoire. Pour un bruit élevé, ce qui correspond en fait à une probabilité d'avoir une déviation angulaire importante par rapport à la direction moyenne, les agents sont très peu alignés pour un même nombre d'itération.

Nous avons ici exactement les mêmes résultats que ceux obtenus par VICSEK et son équipe. Nous sommes également cohérent avec ce que nous avons observé dans la partie 3.6.

Nous avons voulu voir, pour une même densité, le même graphe pour différents nombres d'agents. Ainsi, nous avons augmenté progressivement le nombre d'agents ainsi que la longueur de l'espace considéré en gardant la même densité. Nous avons obtenu ce graphe :

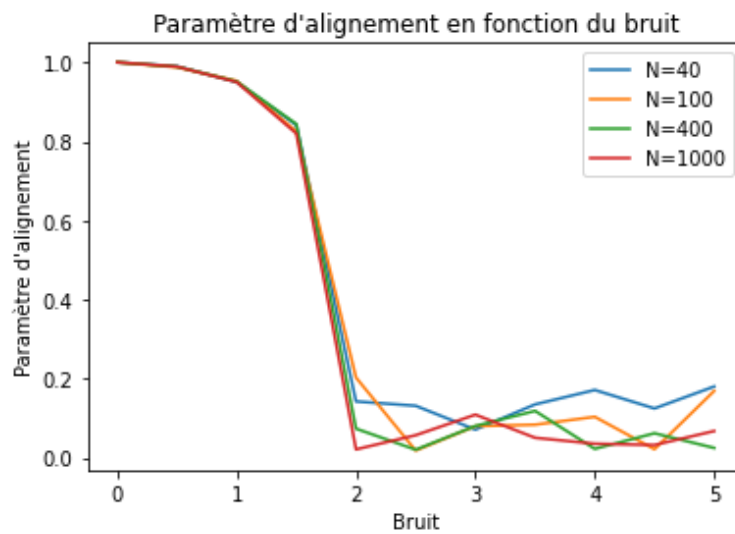


FIGURE 4.2 – Paramètre d'alignement en fonction du bruit pour différents nombre d'agents

Nous remarquons que le paramètre d'alignement diminue drastiquement pour une valeur de bruit passant de 1,8 à 2 (et ce, dans tous les cas que nous avons tracé).

Malgré le fait que l'espace augmente considérablement à chaque fois, les agents ont tendances à s'aligner de moins en moins quand ils sont en grand nombre et dans un espace plus grand. En effet, le paramètre d'alignement est de plus en plus proche de zéro (pour un bruit fixé à 2) quand on augmente le nombre d'agents.

C'est une constatation intéressante. Cela voudrait dire que plus le groupe est nombreux dans un même espace, plus le bruit affecte la cohésion de celui-ci.

Nous en avons d'ailleurs profité pour mesurer l'effet du cône de vision sur le paramètre d'alignement, nous avons tracé ce graphe avec 50 mesures par points sur la figure 4.3.

Nous pouvons ainsi constater que le cône de vision précipite la transition de phase entre un état parfaitement ordonné et un état chaotique. Cela est logique puisque avec l'augmentation du bruit, les agents peuvent prendre des trajectoires de plus en plus aléatoires. Ses voisins peuvent alors sortir du cône de vision, ce qui change radicalement la manière dont l'agent se déplace.

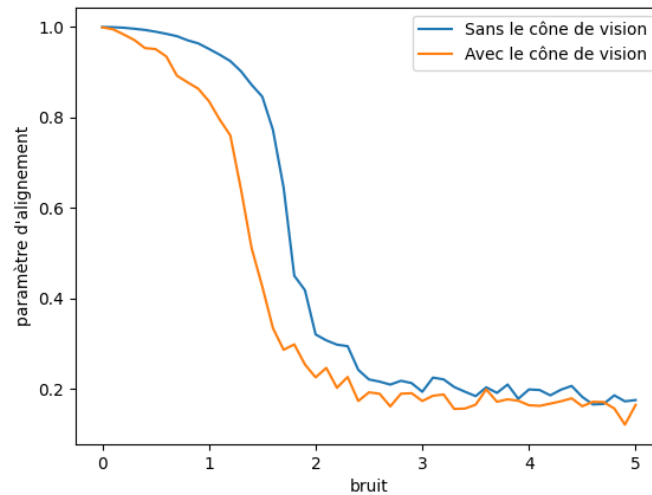


FIGURE 4.3 – Effet du cône de vision sur le paramètre d'alignement avec 50 agents

#### 4.1.2 Paramètre d'alignement en fonction de la densité

VICSEK avait aussi mesuré le paramètre d'alignement en fonction de la densité d'agent dans l'espace (avec un bruit constant). En traçant le paramètre d'alignement en fonction de la densité, nous obtenons pas exactement la même figure que VICSEK même si cela s'en rapproche.

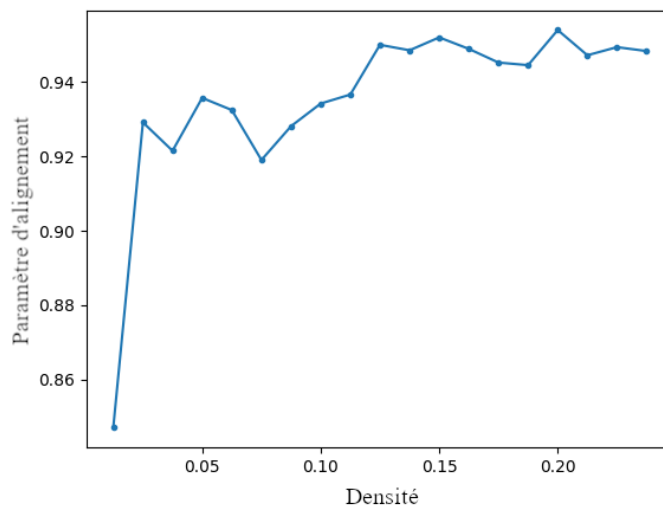


FIGURE 4.4 – Paramètre d'alignement en fonction de la densité d'agent (bruit=1)

Nous pouvons tout de même en déduire que plus le nombre d'agent est élevé dans un même espace (et donc la densité aussi), plus les agents s'aligneront sur la même direction rapidement. Nous pouvons même rajouter qu'à partir d'une valeur de densité d'environ de 0.13 agents par unité d'espace.

Nous n'avons pas clairement identifié quel paramètre différait entre notre modèle et celui de VICSEK. Cependant, il est probable qu'ils s'agisse d'une différence dans la manière de faire évoluer le groupe. En effet, pour chaque point nous repartons de zéro en générant un nouveau groupe alors qu'il est possible que VICSEK garde le même groupe en rajoutant des agents pour faire augmenter la densité, mais même en procédant comme cela, nous ne sommes pas parvenu à retrouver les mêmes courbes.

## 4.2 Au-delà du modèle de VICSEK

### 4.2.1 Création d'un agent leader

Nous avons maintenant voulu nous écarter un peu du modèle de VICSEK, en étudiant un nouveau type de mouvement collectif. Nous avons alors créé un nouveau type d'agent, dit leader, qui influence davantage les agents normaux. Nous pouvons définir à quel point celui-ci à de l'influence (nous pouvons choisir une influence équivalente à  $n$  agents).

Nous avons alors créé le paramètre `agent_type` pour différencier les agents. Ainsi, il nous faut préciser l'indice correspondant à l'agent que l'on veut créer avec `agent_generator`. Il nous faudra ensuite rajouter cet agent au groupe avec la méthode `add_agent`. Nous avons décidé que les agents classiques sont de type 0 et les agents leader de type 1.

Avec l'apparition de ce nouveau type d'agent, nous nous attendions à observer des groupes en forme de pyramide. C'est-à-dire une organisation hiérarchique des agents pour créer un mouvement de groupe. Nous pouvons par exemple observer ce comportement chez certains animaux, notamment lors de la migrations des grues.

Nous avons choisi de représenter l'agent leader en noir.

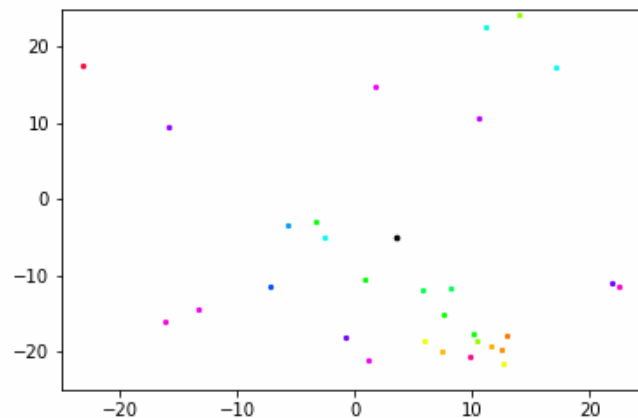


FIGURE 4.5 – Effet observée avec un agent leader

En effet, nous avons observé plusieurs fois une organisation en « triangle ou en arc de cercle », comme une sorte de hiérarchie. L'agent leader produit un mouvement de groupe organisé différemment de ce qu'on a pu observer dans le modèle de VICSEK.

Ainsi, cela montre que le mouvement collectif peut être amené de différentes manières et qu'il n'y a pas de méthode unique pour le déplacement d'un ensemble d'agent.

Nous avons d'ailleurs pu observer quelque chose d'intéressant. En effet, sur une animation, nous pouvons voir un groupe d'agents qui suit l'agent leader pendant un moment. Mais, ce groupe croise un autre groupe d'agents. Les agents qui suivait l'agent leader se sont mis subitement à suivre le nouveau groupe.

Ainsi, nous en déduisons que le groupe a eu plus d'influence que l'agent leader à ce moment là.



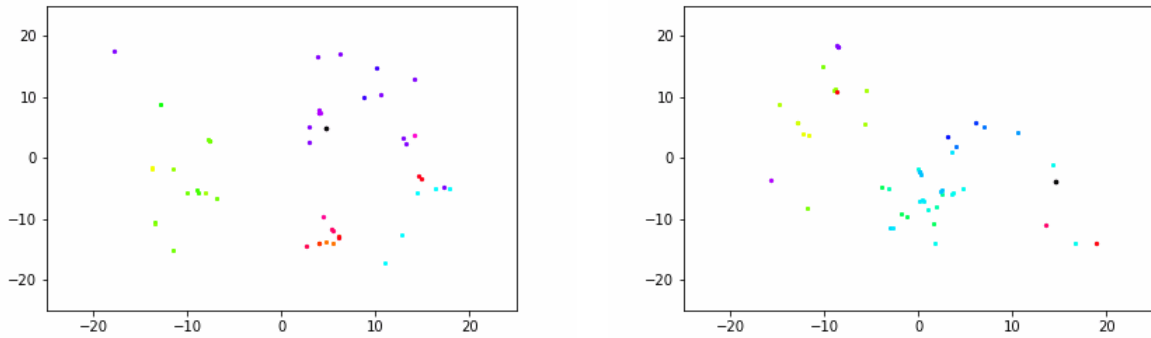


FIGURE 4.6 – Situation intéressante avec un agent leader

### 4.2.2 Mise en place d’une prédation et du paramètre de sensibilité

Nous avons trouvé intéressant de rajouter des agents dit répulsifs, qui joue le rôle de prédateurs. Ces nouveaux agents sont plus rapides que leurs proies. Ainsi, nous avons pu voir comment s’organisent les agents face à la menace.

Nous avons choisit de paramétrer la réaction des agents quand cet agent répulsif est dans la zone de vision indépendamment de l’angle – cela permet de forcer l’agent à fuir, même si l’agent répulsif est derrière. L’agent prendra immédiatement le vecteur de sens opposé à celui vers l’agent répulsif. Nous avons choisit de représenter les agents répulsifs en noir.

Pour créer ce type d’agent, il nous faut préciser `agent_type = 2` dans la méthode `agent_generator`.

Avec cette expérience, nous constatons que le mouvement de groupe est toujours présents avec un seul agent répulsif. Cet effet collectif est même souvent renforcé. Les agents fuient dans la même direction, tout en se servant de leur influence les uns sur les autres.

Cependant, lorsqu’il y a plusieurs agents répulsifs, le mouvement collectif n’est plus observé ou très peu. Ces prédateurs (qui prennent des directions différentes) sèment la panique au sein du groupe qui perd toute cohésion. Chaque agent développe alors un instinct de survie qui le pousse à fuir. La panique et la fuite prennent visiblement le pas sur la cohésion de groupe (cf. figure 4.7).

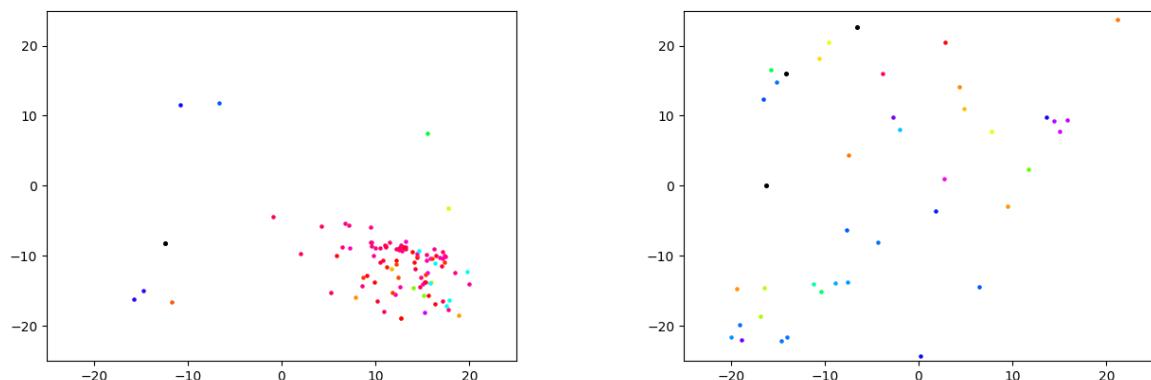


FIGURE 4.7 – Effet d’un (à gauche) ou plusieurs agents répulsifs (à droite) sur le groupe

Nous avons ensuite décidé de rajouter un nouveau paramètre que nous appelons **fear** dans

notre code. Il s'agit de la sensibilité des agents face aux prédateurs, c'est à dire à la peur des agents pour les agents répulsifs.

Ainsi, les agents prennent plus facilement la fuite face aux prédateurs.

### 4.2.3 Système évolutif

En ajoutant des agents répulsifs, nous avons eu l'idée de créer un système de « mort » où les agents touchés par un agent répulsifs sont retirés de la liste des agents et placés dans un autre attribut du groupe.

Cela permet notamment de comparer les caractéristiques des agents morts avec celles des survivants et de voir quels paramètres permettent de survivre. Nous avons créé de nouvelles fonctions appelées `test` qui calculent le pourcentage de survivants.

En générant différents groupes avec `group_generator`, ces fonctions ont 4 groupes à tester pour différents bruits et sensibilité.

En mettant des valeurs limites pour le bruit et la sensibilité aux agents répulsifs et en moyennant les résultats sur 10 mesures, nous obtenons les résultats suivants :

bruit	sensibilité	pourcentage de survivants
1	0	30
0	1	79
1	1	86
0	0	39

Nous voyons bien que statistiquement, les agents ont plus de chance de survivre avec un bruit et une sensibilité à 1. En effet, dans ce cas là, les agents avec une grande sensibilité cherchent à fuir à tout prix le prédateur. De plus, le bruit fort fait que les agents ne cherchent pas à imiter leurs voisins, ce qui ne perturbe pas la fuite qu'il avait entreprise.

Cependant, en enlevant le bruit, nous voyons tout de même que le pourcentage de survivants reste très élevé. La sensibilité est un élément clé pour la survie des agents.

Au contraire, les agents ont peu de chance de survie avec un bruit fort et une sensibilité nulle. En effet, les agents n'ont pas peur de leurs prédateurs, ce qui devient très dangereux. De plus, le bruit élevé, fait qu'il n'y a aucun mouvement de groupe.

Cependant, en retirant le bruit, nous voyons que le pourcentage de survivants est à peine plus haut. Nous revenons à la même conclusion : la sensibilité est un élément clé pour la survie des agents.

### 4.2.4 Ajout d'obstacles

De plus, nous avons voulu voir comment réagissent les agents face à des obstacles. Nous voulions voir le mouvement de groupe est conservé.

Cette approche permet de se placer dans des conditions encore plus réalistes. Par exemple, un banc de poissons peut rencontrer des rochers lors de son déplacement.

Nous avons alors créé des murs. À leur vue, les agents font demi-tour pour éviter l'obstacle.

Pour créer ces murs, il nous faut préciser `agent_type = 3` dans la méthode `agent_generator`.

Nous avons choisis dans un premier temps de représenter un mur au milieu de l'espace considéré. Ces agents de type 3, sont en noir et forment une ligne dans l'animation suivante.

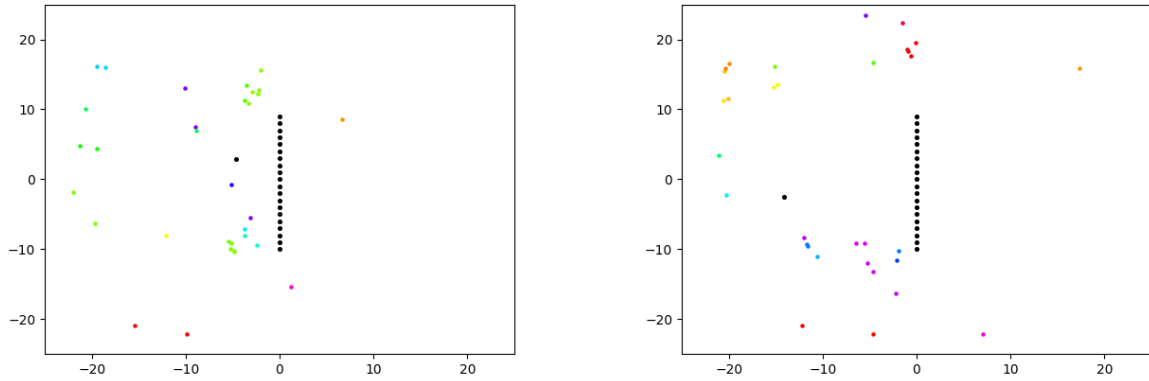


FIGURE 4.8 – Effet d’un obstacle sur le mouvement des agents (arrivée des agents vers le mur à gauche, directions prises par les agents après avoir vu le mur à droite)

Nous voyons bien que les agents rebrousse un peu chemin. En réalité, nous pourrions plutôt nous attendre à ce que le groupe contourne l’obstacle. Il y a tout de même quelques agents qui ont visiblement contourné l’obstacle, mais la majorité du groupe fait demi-tour.

Cependant, il faut noter que l’obstacle est particulièrement imposant, et que cela explique sûrement le comportement du groupe.

Nous avons alors voulu voir l’effet de plusieurs petits obstacles sur un groupe. Pour ce faire, nous avons généré une animation avec six agents de type 3, qui forment un ensemble de petits obstacles répartis sur tout l’espace. Nous obtenons ces images :

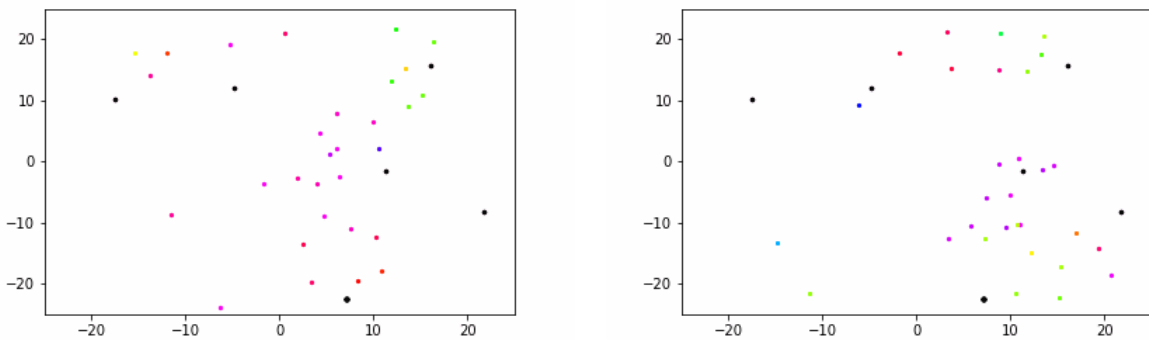


FIGURE 4.9 – Effet de plusieurs petits obstacles sur le mouvement des agents (avant et après l’obstacle)

Dans ce cas là, les agents ne font pas demi-tour mais se fraient un passage entre les petits obstacles. De plus, nous observons que les mouvements de groupes semblent peu affectés par ces six obstacles.

En effet, dans cette animation, les agents contournent les petits obstacles plus facilement. Cela explique le comportement des agents face au grand mur. L’obstacle était trop grand et les agents ont préférés faire demi-tour. Dans ce nouveau cas, ces obstacles sont petits et les agents ont plus d’espace libre pour pouvoir contourner.

La taille de l’obstacle est donc importante et détermine de comportement du groupe.

# Conclusion

Avec la mise en place de ce programme, nous avons réussi à reproduire le modèle de VICSEK de manière numérique. Notre programme implémente le modèle grâce à deux classes composées de plusieurs méthodes qui définissent les paramètres du groupe ainsi que ceux de chaque agent de manière individuelle.

Nous avons pu observer les différents mouvements de groupe et observer le comportement des agents avec différents paramètres essentiels tels que le bruit ou encore le cône de vision. Le bruit, en augmentant, rend le groupe d'agent de plus en plus désordonné. Le cône de vision, nous a permis de mieux voir l'influence des agents entre eux en mettant en avant les agents voisins que l'individu était capable de voir.

De plus, nous avons réussi à retrouver les résultats historique de VICSEK et son équipe concernant le paramètre d'alignement. Nous avons vu l'effet du bruit (densité fixe) ou encore de la densité d'agents (bruit fixe) sur le paramètre d'alignement. Nous confirmons alors ce que nous avons observé précédemment concernant le bruit. Quant à la densité, celle ci en augmentant, rend l'alignement des agents plus facile.

Par la suite, nous avons voulu aller plus loin que le modèle de base en créant de nouveaux types d'agents. Pour commencer, les agents leaders, qui prennent le rôle de guide dans le groupe. Ce comportement nous a fortement fait penser à la migration des grues. Ensuite, nous avons créé une prédation avec l'apparition des agents répulsifs qui sèment la panique dans le groupe. Avec nos statistiques, nous avons déterminer que la meilleure chance de survie est de 86 % avec un bruit et une sensibilité forte. Pour finir, nous avons mis en place un mur qui fait office d'obstacle. Nous voyons ici, que les agents ont tendances à faire demi-tour ou contourner selon la taille de l'obstacle. Cela semble cohérent par rapport à ce que l'on peut observer dans la réalité.

Il serait maintenant intéressant d'étudier les comportements collectifs chez certaines espèces végétales. Nous pensons notamment aux mécanismes de défense mis en place par certains arbres pour faire face à certains dangers (incendies, certains animaux herbivores ou insectes), notamment en communiquant avec leurs voisins. Lorsqu'un arbre est en danger, il peut émettre des signaux chimiques volatils. Ces signaux peuvent être détectés par d'autres arbres de la même espèce qui, à leur tour, peuvent déclencher des mécanismes de défense. Ce mécanisme est très souvent observé chez le séquoia géant (épaisseur de l'écorce face aux feu), ou encore l'eucalyptus (productions de toxines et apparition d'épines).

# Bibliographie

- [1] Modèle de vicsek. [https://en.wikipedia.org/wiki/Vicsek\\_model](https://en.wikipedia.org/wiki/Vicsek_model).
- [2] Tamás vicsek. [https://en.wikipedia.org/wiki/Tamas\\_Vicsek](https://en.wikipedia.org/wiki/Tamas_Vicsek).
- [3] Tamás Vicsek, Andras Czirok, Eshel Ben-Jacob, Inon Cohen, and Shochet. Novel type of phase transition in a system of self-driven particules.
- [4] Matplotlib. <https://matplotlib.org>.

# Annexe A

## Classes et fonctions de générations

---

```
1  """
2  Vicsek -- 1.7.4
3  =====
4  (Alexis Peyroutet et Antoine Royer)
5
6
7  Description
8  -----
9  Implémentation d'un modèle de Vicsek en Python. Permet de simuler
10 ↪ un groupe d'agents en deux ou
11 trois dimensions.
12
13 Cette adaptation se distingue du modèle de Vicsek par plusieurs
14 ↪ aspects :
15 - gestion d'agents attractifs, répulsifs et d'obstacles ;
16 - gestion de l'espace (fermé ou torique) ;
17 - les agents ont un angle de vue limité.
18
19 Exemples
20 -----
21 En important le module comme suit :
22
23 >>> import vicsek as vk
24
25 Pour créer un groupe de 20 agents avec les paramètres par défaut
26 ↪ :
27
28 >>> mon_groupe = vk.group_generator(20)
29
30 Générer une animation de 100 images:
31
32 >>> mon_groupe.compute_animation(100)
33
34 Faire évoluer le groupe de 100 pas sans garder d'images :
35
36 >>> mon_groupe.run(100)
37 """
```

```

35 import math
36 import random
37
38 import matplotlib.pyplot as plt
39 import matplotlib.patches as mpatches
40 import numpy as np
41
42 from matplotlib import animation
43 from matplotlib.collections import PatchCollection
44
45
46 __version__ = "1.7.4"
47
48
49 # Classes
50 class Agent:
51     """
52     Simule un agent avec sa position, sa vitesse, et le bruit
53     ↪ associé qui traduit sa tendance naturelle à suivre le
54     ↪ groupe ou pas.
55
56     Paramètres
57     -----
58     position : np.array
59         Vecteur position, sous forme d'un tableau numpy de deux
60         ↪ ou trois entiers.
61     speed : np.array
62         Direction du vecteur vitesse sous la forme d'un tableau
63         ↪ de trois entiers.
64     velocity : float
65         Norme de la vitesse.
66     noise : float
67         Taux de bruit, qui traduit une déviation aléatoire sur la
68         ↪ direction de la vitesse.
69         Si ``noise=0``, l'agent ne s'écartera jamais de sa
70         ↪ trajectoire.
71         Si ``noise=1``, l'agent aura une trajectoire très bruitée
72     sight : float
73         Distance à laquelle l'agent voit les autres.
74     field_sight : float, optionnel
75         Angle du cône de vision de l'agent en radian.
76     agent_type : int, optionnel
77         Type d'agent
78         0 : agent normal
79         1 : agent répulsif
80         2 : agent leader
81     fear : float, optionnel
82         Sensibilité aux agents répulsifs.
83         Si ``fear=0`` l'agent sera complètement insensible aux
84         ↪ agents répulsifs.

```

```

78         Si ``fear=1`` l'agent aura une sensibilité maximale
79
80     Attributs
81     -----
82     max_velocity : float
83         norme maximale de la vitesse
84     """
85     def __init__(self, position: np.array, speed: np.array,
86         ↪ velocity: float, noise: float, sight: float, field_sight:
87         ↪ float=math.pi/2, agent_type: int=0, fear: float=1):
88         self.position = position.copy()
89         self.speed = speed.copy()
90         self.velocity = velocity
91         self.noise = noise
92         self.sight = sight
93         self.field_sight = field_sight / 2
94         self.agent_type = agent_type
95         self.fear = fear
96
97         self.max_velocity = self.velocity
98         if agent_type == 1:
99             self.max_velocity *= 2
100
101     def __str__(self):
102         """Affiche l'agent avec ses paramètres."""
103         return f"Agent(\n\tposition={list(self.position)}, "\
104             f"\n\tspeed={list(self.speed)}, "\
105             f"\n\tvelocity={self.velocity}, "\
106             f"\n\tnoise={self.noise}, "\
107             f"\n\tsight={self.sight}, "\
108             f"\n\tfield_sight={self.field_sight * 360 / \
109             ↪ math.pi}, "\
110             f"\n\tagent_type={self.agent_type}" \
111             f"\n\tfear={self.fear}\n)"
112
113     __repr__ = __str__
114
115     def __sub__(self, agent):
116         """
117         Retourne la distance entre les deux agents.
118
119         Paramètres
120         -----
121         agent : Agent
122             Un autre agent.
123
124         Signature
125         -----
126         out : float
127             Distance entre self et agent.

```



```

125         """
126         return norm(self.position - agent.position)
127
128     __rsub__ = __sub__
129
130     def __eq__(self, agent):
131         """
132         Teste l'égalité entre deux agents.
133
134         Paramètres
135         -----
136         agent : Agent
137             Un autre agent.
138
139         Signature
140         -----
141         out : bool
142             True si les deux agents ont la même position, False
143             ↪ sinon.
144         """
145         return list(self.position) == list(agent.position)
146
147     def copy(self):
148         """Renvoie une copie profonde de l'agent."""
149         return Agent(self.position.copy(), self.speed.copy(),
150             ↪ self.velocity, self.noise, self.sight,
151             ↪ self.field_sight, self.agent_type, self.fear)
152
153     def get_color(self):
154         """Renvoie la couleur de l'agent en fonction de son
155             ↪ orientation."""
156         if self.agent_type == 3:
157             return (0, 0, 1), 0
158         angle = np.angle(self.speed[0] + 1j * self.speed[1]) % (2
159             ↪ * math.pi)
160         angle = (180 * angle) / math.pi
161         return COLOR_MAP[math.floor(angle) % 360], angle
162
163     def next_step(self, neighbours: list, dim: int, length: int,
164         ↪ step: float=0.5):
165         """
166         Fait évoluer l'agent d'un pas temporel en fonction de ses
167         ↪ voisins.
168
169         Paramètres
170         -----
171         neighbours : list
172             Liste des agents voisins.
173         dim : int
174             Dimension de l'espace.

```

```

168     length : int
169         Longueur caractéristique de l'espace.
170     step : float, optionnel
171         Pas de temps considéré pour les équations
172         ↪ différentielles.
173     """
174     length //= 2
175     average_speed = 0
176     average_velocity = 0
177     nb_neighbours = 0
178
179     for agent in neighbours:
180         if agent.agent_type != 3:
181             average_velocity += agent.velocity
182             nb_neighbours += 1
183
184         if agent.agent_type == 0:
185             if self.agent_type != 1:
186                 average_speed += agent.speed
187             else:
188                 average_speed += (1 / (self - agent)) *
189                 ↪ (agent.position - self.position)
190                 average_velocity += agent.velocity / 4
191
192         elif agent.agent_type == 1:
193             if self.agent_type != 1:
194                 average_speed += self.fear * len(neighbours)
195                 ↪ * (self.position - agent.position)
196             else:
197                 average_speed += agent.speed
198
199         elif agent.agent_type == 2:
200             if self.agent_type != 1:
201                 average_speed += 5 * agent.speed
202             else:
203                 average_speed += (1 / (self - agent)) *
204                 ↪ (agent.position - self.position)
205                 average_velocity += agent.velocity / 4
206
207         elif agent.agent_type == 3:
208             average_speed += len(neighbours) * 100 *
209             ↪ (self.position - agent.position)
210
211     average_speed /= nb_neighbours
212     average_velocity /= nb_neighbours
213     noise_min = - self.noise / 2
214     noise_max = self.noise / 2
215
216     self.position += self.velocity * step * self.speed

```

```

212     self.speed = average_speed + ((noise_max - noise_min) *
    ↪ np.random.random(dim) + noise_min)
213     self.speed /= norm(self.speed)
214
215     if average_velocity > self.max_velocity:
216         average_velocity = self.max_velocity
217     self.velocity = average_velocity
218
219     for i in range(dim):
220         if self.position[i] > length:
221             self.position[i] = -length
222         elif self.position[i] < -length:
223             self.position[i] = length
224
225
226 class Group:
227     """
228     Simule un groupe d'agents, permet de le faire évoluer et de
    ↪ l'afficher.
229
230     Paramètres
231     -----
232     agents : list
233         Liste des agents du groupe.
234     length : int, optionnel
235         Longueur caractéristique de l'espace.
236     dim : int, optionnel
237         Dimension de l'espace considéré (2 ou 3).
238
239     Attributs
240     -----
241     density : float
242         Densité d'agents dans l'espace (nombre agent / longueur
    ↪ ** dimension).
243     dead_agents : list
244         Liste des agents touchés par un agent répulsif.
245     """
246     def __init__(self, agents: list, length: int=50, dim: int=2):
247         self.agents = agents
248         self.dead_agents = []
249         self.nb_agents = len(agents)
250         self.length = length
251
252         if dim not in (2, 3):
253             raise DimensionError("dim must be 2 or 3")
254         self.dimension = dim
255
256         for agent in agents:
257             if len(agent.position) != dim or len(agent.speed) !=
    ↪ dim:

```

```

258         raise DimensionError("dimension of agents don't
           ↪ match")
259
260     self.density = self.nb_agents / (self.length **
           ↪ self.dimension)
261
262     def __getitem__(self, index: int):
263         """
264         Renvoie l'agent d'indice donné.
265
266         Paramètres
267         -----
268         index : int
269             Indice de l'agent.
270
271         Signature
272         -----
273         out : Agent
274             Agent du groupe à l'indice donné.
275         """
276         return self.agents[index]
277
278     def copy(self):
279         """Renvoie une copie profonde du groupe."""
280         return Group([agent.copy() for agent in self.agents],
           ↪ self.length, self.dimension)
281
282     def add_agent(self, agent: Agent):
283         """
284         Permet d'ajouter un agent au groupe.
285         C'est une copie profonde de l'agent qui est ajoutée au
           ↪ groupe.
286
287         Paramètres
288         -----
289         agent : Agent
290             Agent à ajouter.
291         """
292         if len(agent.position) != self.dimension or
           ↪ len(agent.speed) != self.dimension:
293             raise DimensionError("dimension of agent doesn't
           ↪ match")
294         self.agents.append(agent.copy())
295         self.nb_agents += 1
296         self.density = self.nb_agents / (self.length **
           ↪ self.dimension)
297
298     def get_neighbours(self, targeted_agent: Agent, dmin: int,
           ↪ check_field: bool=True,
299                       check_wall: bool=True):

```

```

300     """
301     Calcule les voisins d'un agent en fonction de la distance
        ↪ et de l'angle.

302
303     Paramètres
304     -----
305     targeted_agent : Agent
306         Agent servant de référence pour le calcul de
        ↪ distance.
307     dmin : int
308         Distance minimale à partir de laquelle l'agent sera
        ↪ compté comme voisin.
309     check_field : bool, optionnel
310         Vérification de l'angle de vue. Si
        ↪ ``check_field=False`` les agents voient à 360°.
311     check_wall : bool, optionnel
312         Vérification des murs. Si ``check_wall=False``,
        ↪ l'espace est considéré torique.

313
314     Signature
315     -----
316     agents : list
317         liste des agents voisins
318     """
319     length = self.length / 2
320
321     if self.dimension == 2:
322         wall_agents = [
323             Agent(position=np.array([length,
        ↪ targeted_agent.position[1]]),
324                 speed=np.zeros(2),
325                 velocity=0,
326                 noise=0,
327                 sight=0,
328                 field_sight=0,
329                 agent_type=3),
330             Agent(position=np.array([-length,
        ↪ targeted_agent.position[1]]),
331                 speed=np.zeros(2),
332                 velocity=0,
333                 noise=0,
334                 sight=0,
335                 field_sight=0,
336                 agent_type=3),
337
        ↪ Agent(position=np.array([targeted_agent.position[0],
        ↪ length]),
338             speed=np.zeros(2),
339             velocity=0,
340             noise=0,

```

```

341         sight=0,
342         field_sight=0,
343         agent_type=3),
344
345         ↪ Agent(position=np.array([targeted_agent.position[0],
346         ↪ length]),
347         speed=np.zeros(2),
348         velocity=0,
349         noise=0,
350         sight=0,
351         field_sight=0,
352         agent_type=3),
353     ]
354     else:
355         wall_agents = [
356             Agent(position=np.array([length,
357             ↪ targeted_agent.position[1],
358             ↪ targeted_agent.position[2]]),
359             speed=np.zeros(3),
360             velocity=0,
361             noise=0,
362             sight=0,
363             field_sight=0,
364             agent_type=3),
365             Agent(position=np.array([-length,
366             ↪ targeted_agent.position[1],
367             ↪ targeted_agent.position[2]]),
368             speed=np.zeros(3),
369             velocity=0,
370             noise=0,
371             sight=0,
372             field_sight=0,
373             agent_type=3),
374             ↪ Agent(position=np.array([targeted_agent.position[0],
375             ↪ -length, targeted_agent.position[2]]),
376             speed=np.zeros(3),
377             velocity=0,
378             noise=0,
379             sight=0,
380             field_sight=0,

```

```

381         agent_type=3),
382
383         ↪ Agent(position=np.array([targeted_agent.position[0],
384         ↪ targeted_agent.position[1], -length]),
385         speed=np.zeros(3),
386         velocity=0,
387         noise=0,
388         sight=0,
389         field_sight=0,
390         agent_type=3),
391
392         ↪ Agent(position=np.array([targeted_agent.position[0],
393         ↪ targeted_agent.position[1], length]),
394         speed=np.zeros(3),
395         velocity=0,
396         noise=0,
397         sight=0,
398         field_sight=0,
399         agent_type=3),
400     ]
401
402     agents = []
403     if check_wall:
404         total_agents = self.agents + wall_agents
405     else:
406         total_agents = self.agents
407
408     if not check_field or self.dimension == 3:
409         agents = [agent for agent in total_agents if
410         ↪ (targeted_agent - agent) <= dmin]
411
412     else:
413         dead_index = []
414         for index, agent in enumerate(total_agents):
415             if targeted_agent.agent_type == 1 and not
416             ↪ agent.agent_type in (1, 3) and
417             ↪ (targeted_agent - agent) < self.length / 25
418             ↪ and index < self.nb_agents:
419                 self.dead_agents.append(agent.copy())
420                 dead_index.append(index)
421                 self.nb_agents -= 1
422
423             if agent != targeted_agent:
424                 pos = agent.position -
425                 ↪ targeted_agent.position
426                 angle_spd = np.angle(targeted_agent.speed[0]
427                 ↪ + 1j * targeted_agent.speed[1]) % 2 *
428                 ↪ math.pi
429                 angle_pos = np.angle(pos[0] + 1j * pos[1]) %
430                 ↪ (2 * math.pi)

```

```

419         if agent.agent_type != 3:
420             if (targeted_agent - agent) <= dmin and
421                 ↪ (agent.agent_type == 1 or
422                 ↪ abs(angle_spd - angle_pos) <=
423                 ↪ targeted_agent.field_sight):
424                 agents.append(agent)
425             elif (targeted_agent - agent) <= self.length
426                 ↪ / 25:
427                 agents.append(agent)
428
429         else: agents.append(agent)
430
431     for index in dead_index:
432         self.agents.pop(index)
433
434     return agents
435
436 def get_agents_arguments(self):
437     """Retourne un tuple de tableaux numpy contenant les
438     ↪ positions et les vitesses de tous les agents du
439     ↪ groupe."""
440     positions = np.zeros((self.nb_agents, self.dimension))
441     speeds = np.zeros((self.nb_agents, self.dimension))
442
443     for index, agent in enumerate(self.agents):
444         positions[index] = agent.position
445         speeds[index] = agent.velocity * agent.speed
446
447     return positions, speeds
448
449 def compute_figure(self):
450     """Génère une figure matplotlib avec le groupe d'agents
451     ↪ sous forme d'un nuage de points en deux ou trois
452     ↪ dimensions."""
453     fig = plt.figure()
454     if self.dimension == 2:
455         plot_axes = plt.axes()
456         sight_wedges = []
457
458         for agent in self.agents:
459             color, dir_angle = agent.get_color()
460             sight_angle = (180 * agent.field_sight) / math.pi
461
462             if agent.agent_type:
463                 size, color = 7, (0, 0, 0)
464             else:
465                 size = 5
466             plt.scatter(agent.position[0], agent.position[1],
467                 ↪ s=size, color=color)

```



```

460         wedge = mpatches.Wedge((agent.position[0],
    ↪     agent.position[1]), agent.sight,
461             dir_angle + 360 - sight_angle,
    ↪     dir_angle + sight_angle,
    ↪     ec="black")
462     sight_wedges.append(wedge)
463
464
    ↪     plot_axes.add_collection(PatchCollection(sight_wedges,
    ↪     alpha=0.3))
465
466     plot_axes.axes.set_xlim(-self.length / 2, self.length
    ↪     / 2)
467     plot_axes.axes.set_ylim(-self.length / 2, self.length
    ↪     / 2)
468
469     return fig
470
471     def show(self):
472         """Affiche le groupe d'agent."""
473         self.compute_figure()
474         plt.show()
475
476     def compute_animation(self, frames: int=20, interval:
    ↪     int=100, filename: str="vicsek", check_field: bool=True,
    ↪     check_wall: bool=False, sight: bool=False, step:
    ↪     float=0.5):
477         """
478         Génère une animation.
479
480         Paramètres
481         -----
482         frames : int, optionnel
483             Nombre d'images voulues dans l'animation.
484         interval : int, optionnel
485             Intervale temporel entre deux frames de l'animation
    ↪         en ms.
486         filename : str, optionnel
487             Nom du fichier de sortie GIF.
488         check_field : bool, optionnel
489             Vérification de l'angle de vue. Si
    ↪         ``check_field=False`` les agents voient à 360°.
490         check_wall : bool, optionnel
491             Vérification des murs. Si ``check_wall=False``,
    ↪         l'espace est considéré torique.
492         sight : bool, optionnel
493             Affichage du cône de vision de chaque agent.
494         step : float, optionnel
495             Pas temporel pris pour les équations différentielles.
496         """

```

```

497     def aux(frame_index, plot_axes, sight: bool=True):
498         progress_bar(frame_index, frames,
499             ↪ finished="exportation GIF en cours")
500
501         sight_wedges = []
502         plot_data = []
503         size = 5
504
505         if self.dimension == 2:
506             for agent in self.agents:
507                 if agent.agent_type != 3:
508                     agent.next_step(
509                         self.get_neighbours(agent,
510                             ↪ agent.sight, check_field,
511                             ↪ check_wall),
512                         self.dimension, self.length,
513                         ↪ step)
514
515                 color, dir_angle = agent.get_color()
516                 sight_angle = (180 * agent.field_sight) /
517                     ↪ math.pi
518
519                 if agent.agent_type:
520                     size, color = 7, (0, 0, 0)
521                 else:
522                     size = 5
523
524                 ↪ plot_data.append(plot_axes.scatter(agent.position[0],
525                 ↪ agent.position[1],
526                     s=size, color=color))
527
528                 if sight or agent.agent_type == 1:
529                     wedge =
530                         ↪ mpatches.Wedge((agent.position[0],
531                         ↪ agent.position[1]), agent.sight,
532                             dir_angle + 360 - sight_angle,
533                             ↪ dir_angle + sight_angle,
534                             ec="none")
535                     sight_wedges.append(wedge)
536
537                 ↪ plot_data.append(plot_axes.add_collection(PatchCollect
538                 ↪ alpha=0.3)))
539
540         else:
541             for agent in self.agents:
542                 if agent.agent_type != 3:

```

533

```

    ↪ agent.next_step(self.get_neighbours(agent,
    ↪ agent.sight, check_field),
    self.dimension, self.length,
    ↪ step)

```

534

```

sight_angle = (180 * agent.field_sight) /
    ↪ math.pi

```

536

```

if agent.agent_type:
    size, color = 7, (1, 0, 0)
else:
    size, color = 5, (0, 0, 0)

```

537

538

539

540

541

542

```

    ↪ plot_data.append(plot_axes.scatter(agent.position[0],
    ↪ agent.position[1], agent.position[2],
    ↪ s=size, color=color))

```

543

```

    ↪ plot_data.append(plot_axes.quiver(agent.position[0],
    ↪ agent.position[1],
    agent.position[2], agent.velocity *
    ↪ agent.speed[0],
    agent.velocity * agent.speed[1],
    ↪ agent.velocity * agent.speed[2],
    color=color))

```

544

545

546

547

548

```

    return plot_data

```

549

```

images = []

```

551

```

fig = plt.figure()

```

552

```

if self.dimension == 2:

```

553

```

    plot_axes = plt.axes()
    plot_axes.axes.set_xlim(-self.length / 2, self.length
    ↪ / 2)

```

554

555

```

    plot_axes.axes.set_ylim(-self.length / 2, self.length
    ↪ / 2)

```

556

```

else:

```

557

```

    plot_axes = plt.axes(projection="3d")
    plot_axes.axes.set_xlim3d(-self.length / 2,
    ↪ self.length / 2)
    plot_axes.axes.set_ylim3d(-self.length / 2,
    ↪ self.length / 2)
    plot_axes.axes.set_zlim3d(-self.length / 2,
    ↪ self.length / 2)

```

558

559

560

561

562

```

for findex in range(frames):
    plot_data = aux(findex, plot_axes, sight)
    images.append(plot_data)

```

563

564

565

566

```

567     ani = animation.ArtistAnimation(fig, images,
    ↪     interval=interval)
568     ani.save(filename + ".gif")
569
570     def run(self, steps: int=20, check_field: bool=True,
    ↪     check_wall: bool=True, step: float=0.5):
571         """
572         Fait avancer le groupe d'agent sans gérer d'animations.
573
574         Paramètres
575         -----
576         steps : int, optionnel
577             Nombre de pas dont il faut faire avancer le groupe.
578         check_field : bool, optionnel
579             Vérification de l'angle de vue. Si
    ↪             ``check_field=False`` les agents voient à 360°.
580         check_wall : bool, optionnel
581             Vérification des murs. Si ``check_wall=False``,
    ↪             l'espace est considéré torique.
582         step : float, optionnel
583             Pas temporel pris pour les équations différentielles.
584         """
585         for index in range(steps):
586             progress_bar(index, steps)
587             for agent in self.agents:
588                 if agent.agent_type != 3:
589                     agent.next_step(
590                         self.get_neighbours(agent,
    ↪                         agent.sight, check_field,
    ↪                         check_wall),
591                         self.dimension, self.length,
    ↪                         step)
592
593     def order_parameter(self):
594         """Renvoie le paramètre d'alignement."""
595         speed = np.zeros(self.dimension)
596         velocity = 0
597         for agent in self.agents:
598             agent_speed = agent.velocity * agent.speed
599
600             speed += agent_speed
601             velocity += norm(agent_speed)
602
603         return (1 / velocity) * norm(speed)
604
605
606     class DimensionError(Exception):
607         """Erreur de dimension."""
608
609

```

```

610 # Fonctions
611 def rarray(dim: int, minimum: float, maximum: float):
612     """
613     Retourne un tableau numpy de taille donnée rempli de nombres
614     ↪ aléatoire pris entre les bornes
615     communiquées.
616
617     Paramètres
618     -----
619     dim : int
620         Dimension du tableau.
621     minimum : float
622         Valeur minimum du tableau.
623     maximum : float
624         Valeur maximum du tableau.
625
626     Signature
627     -----
628     out : np.array
629         Tableau numpy de nombres aléatoires.
630     """
631     return minimum + (maximum - minimum) * np.random.random(dim)
632
633 def agent_generator(position: tuple=(-25, 25), speed: tuple=(-2,
634     ↪ 2), noise: tuple=(0, 1), sight: tuple=(5, 10), field_sight:
635     ↪ tuple=(math.pi/4, math.pi/2), agent_type: int=0, fear:
636     ↪ tuple=(0, 1), dim: int=2):
637     """
638     Retourne un agent généré aléatoirement.
639
640     Paramètres
641     -----
642     position : tuple, optionnel
643         Valeurs limites de la position.
644     speed : tuple, optionnel
645         Valeurs limites de la vitesse.
646     noise : tuple, optionnel
647         Valeurs limites du bruit de l'agent.
648     sight : tuple, optionnel
649         Valeurs limites de la portée de la vue des agents.
650     field_sight : tuple, optionnel
651         Valeurs limites du champ de vision des agents.
652     agent_type : int, optionnel
653         Type de l'agent
654         0 : agent normal
655         1 : agent répulsif
656         2 : agent leader
657         (3 : mur)
658     fear : tuple, optionnel

```

```

656     Valeurs limites de la peur de l'agent aux agents
        ↪ répulsifs.
657 dim : int, optionnel
658     Dimension de l'espace, peut être 2 ou 3.
659
660 Signature
661 -----
662 agent : Agent
663     Agent généré dans la limite des paramètres donnés.
664 """
665 if noise == -1:
666     noise = random.random()
667 if fear == -1:
668     fear = random.random()
669 agent = Agent(
670     position=rarray(dim, position[0], position[1]),
671     speed=np.zeros(dim),
672     velocity=0,
673     noise=(noise[1] - noise[0]) * random.random() + noise[0],
674     sight=(sight[1] - sight[0]) * random.random() + sight[0],
675     field_sight=(field_sight[1] - field_sight[0]) *
        ↪ random.random() + field_sight[0],
676     agent_type=agent_type,
677     fear=(fear[1] - fear[0]) * random.random() + fear[0]
678 )
679
680 velocity = 0
681 while velocity < 1e-3:
682     agent.speed = rarray(dim, speed[0], speed[1])
683     velocity = norm(agent.speed)
684
685 agent.speed /= velocity
686 agent.velocity = velocity
687 return agent.copy()
688
689
690 def group_generator(nb_agents: int, position: tuple=(-25, 25),
        ↪ speed: tuple=(-2, 2), noise: tuple=(0, 1), sight: tuple=(5,
        ↪ 10), field_sight: tuple=(math.pi/4, math.pi/2), fear:
        ↪ tuple=(0, 1), length: int=50, dim: int=2):
691     """
692     Retourne un groupe d'agents normaux générés aléatoirement
        ↪ dans les limites données.
693
694 Paramètres
695 -----
696 nb_agents : int
697     Nombre d'agents à générer pour le groupe.
698 position : tuple, optionnel
699     Valeurs limites de la position.

```

```

700     speed : tuple, optionnel
701         Valeurs limites de la vitesse.
702     noise : tuple, optionnel
703         Valeurs limites du bruit de l'agent.
704     sight : tuple, optionnel
705         Valeurs limites de la portée de la vue des agents.
706     field_sight : tuple, optionnel
707         Valeurs limites du champ de vision des agents.
708     fear : tuple, optionnel
709         Valeurs limites de la peur de l'agent aux agents
710         ↪ répulsifs.
711     length : int, optionnel
712         Longueur caractéristique de l'espace.
713     dim : int, optionnel
714         Dimension de l'espace, peut être 2 ou 3.
715
716     Signature
717     -----
718     out : Group
719         Groupe contenant les agents générés dans les limites
720         ↪ données et avec les paramètres de
721         longueur et de dimension donnés.
722     """
723     agents = [agent_generator(
724         position=position,
725         speed=speed,
726         noise=noise,
727         sight=sight,
728         field_sight=field_sight,
729         fear=fear, dim=dim)
730         for _ in range(nb_agents)]
731     return Group(agents, length=length, dim=dim)
732
733 def norm(vect: np.array):
734     """
735     Renvoie la norme du vecteur passé en argument.
736
737     Paramètres
738     -----
739     vect : np.array
740         Vecteur n-dimensionnel.
741
742     Signature
743     -----
744     out : float
745         Norme du vecteur.
746     """
747     return math.sqrt(sum(vect ** 2))

```

```

748
749 def get_colors():
750     """Retourne une liste de couleur indexée sur l'angle avec
751     ↪ l'horizontale ascendante."""
752     color_map = []
753     red, green, blue = 255, 0, 0
754     for angle in range(360):
755         if (angle // 60) == 0:
756             green += 4.25
757         elif (angle // 60) == 1:
758             red -= 4.25
759         elif (angle // 60) == 2:
760             blue += 4.25
761         elif (angle // 60) == 3:
762             green -= 4.25
763         elif (angle // 60) == 4:
764             red += 4.25
765         elif (angle // 60) == 5:
766             blue -= 4.25
767         color_map.append((red / 255, green / 255, blue / 255))
768     return color_map
769
770 def progress_bar(iteration: int, total: int, finished: str=""):
771     """
772     Affiche une barre de progression.
773
774     Paramètres
775     -----
776     iteration : int
777         Itération courante à afficher.
778     total: int
779         Nombre total d'itération sur la barre
780     finished : str, optionnel
781         Texte à afficher une fois la barre complète.
782     """
783     iteration += 1
784     completed_length = math.floor(75 * iteration / total)
785     track = "#" * completed_length + " " * (75 -
786     ↪ completed_length)
787     print(f"[{track}] {math.floor(100 * iteration / total)}%",
788     ↪ end="\r")
789
790     if iteration == total:
791         if finished:
792             print("\n" + finished)
793         else:
794             print()

```



```
795 def stat(agents):
796     """
797     Affiche des statistiques sur une liste d'agents.
798     - bruit moyen
799     - peur moyenne
800     - vitesse moyenne
801     """
802     noise, fear, velocity = 0, 0, 0
803     nb_agents = 0
804
805     for agent in agents:
806         if agent.agent_type in (0, 2):
807             nb_agents += 1
808             noise += agent.noise
809             fear += agent.fear
810             velocity += agent.max_velocity
811
812     noise /= nb_agents
813     fear /= nb_agents
814     velocity /= nb_agents
815
816     print("noise :", noise)
817     print("fear :", fear)
818     print("vitesse :", velocity)
819
820
821 # Constantes
822 COLOR_MAP = get_colors()
```

---

# Annexe B

## Code d'utilisation

---

```
1  """
2  Vicsek stat
3  =====
4
5  Description
6  -----
7  Permet de faire des statistiques à partir du modèle de Vicsek et
8  ↪ avec différents paramètres.
9  """
10 import numpy as np
11 import vicsek as vi
12
13 def op_noise(check_field: bool=False, check_wall: bool=False):
14     """
15     Calcule le paramètre d'alignement pour différentes valeurs de
16     ↪ bruits et renvoie un tuple de la forme (bruit, paramètre
17     ↪ d'alignement).
18
19     Paramètres
20     -----
21     check_field : bool, optionnel
22         Prise en compte de l'angle de vue des agents.
23     check_wall : bool, optionnel
24         Prise en compte des murs.
25
26     Signature
27     -----
28     out : tuple
29         Tuple de deux listes contenant respectivement les valeurs
30         ↪ de bruit et du paramètres d'alignement.
31     """
32     order_p = []
33     noises = np.arange(0, 5.5, 0.10)
34     for noise in noises:
35         op_temp = 0
36         # for _ in range(5):
```

```

34     grp = vi.group_generator(40, position=(-1.5, 1.5),
    ↪ speed=(-1, 1), noise=(noise, noise), length=3.1)
35     grp.run(100, check_field=check_field,
    ↪ check_wall=check_wall, step=0.5)
36     op_temp += grp.order_parameter()
37
38     order_p.append(op_temp)
39     print()
40     return list(noises), order_p
41
42
43 def op_density():
44     """
45     Calcule le paramètre d'alignement pour différentes densités
    ↪ et renvoie un tuple de la forme (densité, paramètre
    ↪ d'alignement).
46
47     Signature
48     -----
49     out: tuple
50         Tuple de deux listes contenant respectivement les valeurs
    ↪ de densité et du paramètres d'alignement.
51     """
52     order_p = []
53     density = []
54     grp = vi.group_generator(2, position=(-0.5, 0.5),
    ↪ speed=(-0.5, 0.5), noise=(1.5, 1.5), length=1)
55
56     for _ in range(100):
57         for _ in range(1):
58             agent = vi.agent_generator(position=(-0.5, 0.5),
    ↪ speed=(-0.5, 0.5), noise=(1.5, 1.5))
59             grp.add_agent(agent)
60
61             grp.run(10, check_field=False, check_wall=False,
    ↪ step=0.25)
62
63             order_p.append(grp.order_parameter())
64             density.append(grp.density)
65         print()
66     return density, order_p
67
68
69 def neutral_alignment():
70     """
71     Retourne le paramètre d'alignement en fonction de la densité
    ↪ sans itérer le modèle.
72
73     Signature
74     -----

```

```

75     out : tuple
76         Tuple de deux listes contenant respectivement les valeurs
77         ↪ de densité et du paramètres d'alignement. """
78     def get_op():
79         order_p = []
80         density = []
81         for numb in np.arange(5, 100, 5):
82             op_temp = 0
83             density_temp = 0
84             for _ in range(5):
85                 grp = vi.group_generator(numb, position=(-10,
86                 ↪ 10), speed=(-1, 1), noise=(1.5, 1.5),
87                 ↪ length=20)
88                 op_temp += grp.order_parameter()
89                 density_temp += grp.density
90
91                 order_p.append(op_temp / 5)
92                 density.append(density_temp / 5)
93             return density, order_p
94
95     runs = np.array([get_op() for _ in range(50)])
96     density = runs[:, 0]
97     order_p = runs[:, 1]
98
99     avg_op = []
100     for index in range(19):
101         avg_op.append(sum(order_p[:, index]) / 50)
102
103     return density[0], avg_op
104
105 def stat(fct, *args, iteration: int=10):
106     """
107     Permet de faire des essais sur un plus grand nombre de cas et
108     ↪ de moyenner les résultats.
109
110     Paramètres
111     -----
112     fct : builtin_function_or_method
113         Fonction sur laquelle il faut faire les essais.
114         Cette fonction doit renvoyer un tuple de deux listes.
115     *args : optionnel
116         Arguments de `fct`.
117     iteration : int, optionnel
118         Nobmre de tests à effectuer.
119     Signature
120     -----
121     out : tuple
122         Tuple de deux listes correspondants aux moyennes des
123         ↪ résultats renvoyés par `fct`.

```

```

120     """
121     runs = []
122     for i in range(iteration):
123         print(f"{i+1} / {iteration}")
124         runs.append(fct(*args))
125
126     runs = np.array(runs)
127     x_axis = runs[:, 0][0]
128     y_axis = runs[:, 1]
129
130     avg_y = []
131     for index in range(len(x_axis)):
132         avg_y.append(sum(y_axis[:, index]) / len(y_axis[:,
133             ↪ index]))
134
135     return x_axis, avg_y
136
137 def predation():
138     """Calcule le pourcentage de survivant dans des cas extrêmes
139     ↪ de bruit et de sensibilité avec 500 itérations."""
140     group_1 = vi.group_generator(50, noise=(0, 0), fear=(0, 0))
141     for _ in range(2):
142         group_1.add_agent(vi.agent_generator(speed=(-3, 3),
143             ↪ noise=(0.25, 0.25), agent_type=1))
144
145     group_2 = group_1.copy()
146     group_3 = group_1.copy()
147     group_4 = group_1.copy()
148     for i in range(50):
149         group_1[i].noise, group_1[i].fear = 1, 0
150         group_2[i].noise, group_2[i].fear = 0, 1
151         group_3[i].noise, group_3[i].fear = 1, 1
152         group_4[i].noise, group_4[i].fear = 0, 0
153
154     group_1.run(500)
155     group_2.run(500)
156     group_3.run(500)
157     group_4.run(500)
158
159     return len(group_1.dead_agents) / 50,
160     ↪ len(group_2.dead_agents) / 50, len(group_3.dead_agents) /
161     ↪ 50, len(group_4.dead_agents) / 50
162
163 def predation_stat():
164     """Statistiques sur 10 lancement de `predation`."""
165     rslt = [-1, -1, -1, -1]
166     for i in range(10):
167         print(f"{i + 1}/10")

```

```
165     grps = predation()
166     for i in range(4):
167         if rslt[i] != -1:
168             rslt[i] = (rslt[i] + grps[i]) / 2
169         else:
170             rslt[i] = grps[i]
171
172     print("noise | fear | % survivants")
173     print("1      | 0      |", round(1 - rslt[0], 2))
174     print("0      | 1      |", round(1 - rslt[1], 2))
175     print("1      | 1      |", round(1 - rslt[2], 2))
176     print("0      | 0      |", round(1 - rslt[3], 2))
```

---