Projet d'application final

Multi-scale tracking of collective movement

Auteurs :
Calves Delhio
Carreau Matthieu
Ducottet Rémi
Triana Paul

Encadré par :
Dr. Ada Diaconescu

1er juillet 2022

Résumé

Au cours du projet d'application final (PAF) qui se déroule à la fin de la première année du Cycle Ingénieur de Télécom Paris, nous avons travaillé sur une simulation de groupes d'agents dont le comportement individuel est régi par celui des agents voisins et par l'environnement. Pour ce faire, nous avons utilisé l'environnement de développement Processing, qui est basé sur la plateforme Java, mais qui a l'avantage d'être particulièrement adapté à la création graphique. En effet, nous sommes partis d'une base existante en récupérant le code de Daniel Shiffman sur le site web de Processing, qui simule le déplacement d'individus au sein d'une nuée d'oiseaux, nous avons entrepris de saisir les mécanismes de cette simulation, puis nous l'avons enrichie. L'objectif de notre travail était notamment d'être capable de détecter les différents groupes d'agents, de suivre leur mouvement et de prédire leur trajectoire en se basant sur leurs mouvements passés.

1 Introduction

On parle de mouvements collectifs lorsque plusieurs entités présentant une proximité spatiale changent simultanément de position. L'étude de ces mouvements peut s'avérer utile car ils correspondent à des situations réelles : déplacement d'une nuée d'oiseaux, d'une foule dans une gare, ou encore celui d'un banc de poissons. Ce projet de suivi de mouvements collectifs est donc basé sur une simulation informatique contenant plusieurs agents interagissant dans un environnement selon des règles bien définies, et son objectif est pluriel :

— détecter les différents collectifs d'agents en identifiant ses membres
— être capable de détecter la fusion ou la séparation de ces groupes en groupes plus petits
— assurer le suivi de la position de ces groupes pour en dégager la trajectoire
— prédire la trajectoire à venir de ces groupes d'agents
— exporter et analyser a posteriori la trajectoire réalisée, évaluer la pertinence de la prédiction

Pour ce faire, nous nous sommes basés sur une simulation pré-existante, intitulée "Flocking" de Daniel Shiffman, dont nous avons tenté de comprendre les mécanismes. Ensuite, nous avons enrichi cette simulation de manière incrémentale, en ajoutant des fonctionnalités peu à peu.

2 Fonctionnement de la simulation d'agents

La simulation dans sa forme initiale est constitué d'un espace bidimensionnel périodique ou fermé représenté par une fenêtre finie et d'agents pouvant se déplacer et subir des forces que nous établirons par la suite. Dans le cas de l'espace périodique, lorsque l'agent arrive à la frontière de cet espace, il est reporté au côté opposé en gardant le même vecteur vitesse. En notant x_te la coordonnée à un instant t de l'agent selon l'axe $\vec{x}$ dans une fenêtre de longueur n, cette propriété se traduit par : si $x_t > n$, alors $x_{t+1} = 0$ et si $x_t < 0$, alors $x_{t+1} = n$. Dans le cas d'un espace fermé, les bords de la fenêtre sont infranchissables par les agents, ainsi en notant $v_{x,t}$ la coordonnée vitesse selon $\vec{x}$ à un instant t, cette propriété devient : si $x_t > n$ ou $x_t < 0$, alors $v_{x,t+1} = -v_{x,t}$. Dans la suite, comme nous nous intéresserons à l'étude des groupes d'agents et notamment leur interaction avec leur environnement, nous utiliserons le modèle fermé.

Notre simulation est basée sur celle proposée par Daniel Shiffman [1]. Cette première version propose un fonctionnement avec les trois forces principales (alignement, cohésion et séparation) mais nous avons décider de proposer des enrichissements à cette base algorithmique en ajoutant la présence d'obstacle qui rajoutent donc une nouvelle force dont dépend le mouvement des agents.

Cette simulation présente des similitudes avec des phénomènes naturels et permet de reproduire des mouvements collectifs que l'on peut observer chez différentes espèces animales, notamment dans les nuées d'oiseaux ou dans les bancs de poissons.

2.1 L'agent

L'agent que nous utilisons dans notre simulation est représenté par un vecteur position, vitesse et accélération. La représentation que nous avons choisie est montrée à la Figure 1. L'orientation de la tête triangulaire permet de représenter sa direction de vitesse et la traînée permet de visualiser ses positions du passé proche. Nous verrons également par la suite que sa couleur montre son appartenance à un groupe d'agents que nous déterminons grâce à un algorithme de *clustering* (regroupement en français).

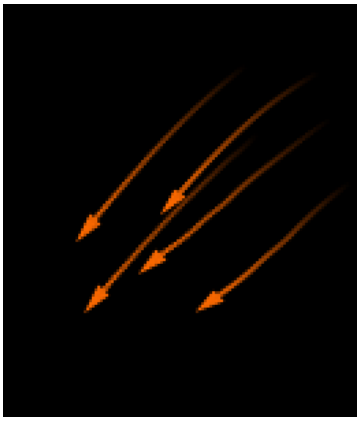


FIGURE 1 – Exemple de plusieurs agents

Le principe de cette simulation est que chaque agent interagit, à travers diverses forces attractives et répulsives, avec les agents de son entourage. Il est donc important de définir ce que l'algorithme comprend par son entourage. La première idée et de ne considérer que les agents se trouvant à une distance inférieure à une longueur prédéfinie d. Nous verrons dans la suite comment l'implémentation d'une partition en grille permet d'optimiser cette étape.

2.2 Forces et mouvement

Établissons dans un premier temps les principes qui construisent nos trois lois de forces initiales.

2.2.1 Alignement

Cette première force contraint un agent à s'aligner avec les agents de son entourage. On commence par calculer la force des sommes des vitesses de son entourage.

input : $agent$
output: $\vec{sum}$
1 $count \leftarrow 0$
2 $\vec{sum} \leftarrow \vec{0}$
3 **for** a *in* $Voisins$ **do**
4 $\quad \vec{v} \leftarrow \vec{v}_a$ `// ` $\vec{v}_a$ `est le vecteur vitesse de` a
5 $\quad \vec{sum} \leftarrow \vec{sum} + \vec{v}$
6 $\quad count \leftarrow count + 1$
7 **end**
8 $\vec{sum} \leftarrow \frac{1}{count}\vec{sum}$

Algorithme 1: Force d'alignement

On obtient donc la vitesse moyenne correspondant aux voisins de l'agent. Cherchant à s'aligner avec ces derniers, cette vitesse est donc la vitesse désirée par l'agent, autrement dit celle qu'il cherche à atteindre.

2.2.2 Séparation

La deuxième force, appelée force de séparation empêche un agent de trop s'approcher des autres agents. Dans un premier temps, on définit un distance de séparation avec les autres agents que l'on souhaite obtenir. Ensuite, pour chaque agent, s'il est trop proche d'un autre agent, on crée un vecteur unitaire à l'opposé de cet agent extérieur par rapport à l'agent étudié, pour qui on calcule la force. On multiplie ensuite par l'inverse de la distance qui sépare ces deux agents.

input : $agent$
output: $\vec{sum}$

$\vec{sum} \leftarrow \vec{0}$
for a *in* $Voisins$ **do**
  $\vec{d} \leftarrow \vec{dist}(a, agent)$
  if $\|\vec{d}\| < desiredseperation$ **then**
    $\vec{sum} \leftarrow \vec{sum} + \frac{1}{\|\vec{d}\|^2}\vec{d}$ `// on normalise le vecteur puis on le divise par la`
    $count \leftarrow count + 1$ `// distance`
  end
end
$\vec{sum} \leftarrow \frac{1}{count}\vec{sum}$

Algorithme 2: Force de séparation

2.2.3 Cohésion

Enfin, la dernière force originelle de l'algorithme est celle de la cohésion, cette force entraîne l'agent à se diriger vers le centre du groupe de voisins dans lequel il se trouve.

input : $agent$
output: $\vec{sum}$

$count \leftarrow 0$
$\vec{sum} \leftarrow \vec{0}$
for a *in* $Voisins$ **do**
  $\vec{d} \leftarrow \vec{dist}(agent, a)$ `// dist renvoie le vecteur de agent à a`
  $\vec{sum} \leftarrow \vec{sum} + \vec{d}$
  $count \leftarrow count + 1$
end
$\vec{sum} \leftarrow \frac{1}{count}\vec{sum}$

Algorithme 3: Force de cohésion

En sortie de cette algorithme, on obtient la moyenne des vecteurs dirigés vers les autres agents, c'est-à-dire un vecteur dirigé vers le barycentre des voisins.

2.3 Calcul de force

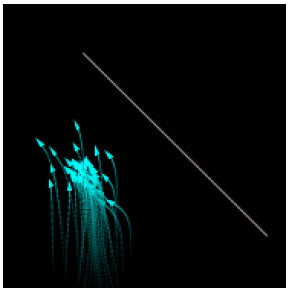
En sortie des trois algorithmes précédents nous obtenons un vecteur mais qui ne correspond pas encore à notre force mais à un vecteur homogène à une vitesse. C'est plus précisément une vitesse désirée. Ainsi en prenant la différence de ce vecteur vitesse désiré et du vecteur vitesse, on obtient l'ajout vitesse qu'il nous faudrait. On a donc : $\vec{v_{t+dt}} - \vec{v_t} = \vec{v_{désirée}} - \vec{v_{actuelle}} = \vec{a_t} * dt = \vec{force} * dt$. On choisira dans la suite $dt = 1$. On peut donc obtenir la force correspondante de la manière suivante : (voir algorithme 4)

input : $agent, \vec{sum}$: la sortie d'un de trois algorithmes précédents
output: $\vec{f}$: le vecteur force correspondant

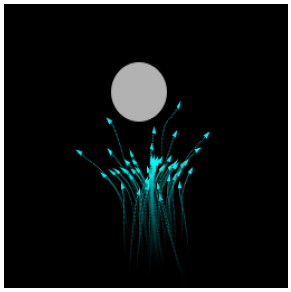
$\vec{sum} \leftarrow \frac{1}{\|\vec{sum}\|}\vec{sum}$
$\vec{sum} \leftarrow maxSpeed * \vec{sum}$
$\vec{f} \leftarrow \vec{sum} - \vec{v_{agent}}$
$\vec{f} \leftarrow \vec{f}.limit(maxForce)$

Algorithme 4: Calcul de forces

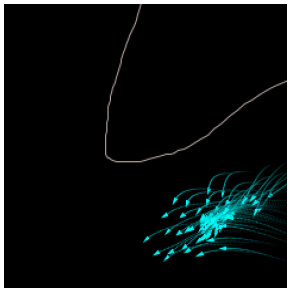
Les constantes $maxSpeed$ et $maxForce$ correspondent respectivement au plafond pour la vitesse et pour la force.



(a) Interaction avec un *Wall*



(b) Interaction avec un cercle



(c) Interaction avec une forme

FIGURE 2 – Interaction avec trois obstacles différents

2.4 Les obstacles

Les obstacles représentent des objets qui ont une position spatiale et qui exercent une nouvelle force sur l'agent. On a implémenté deux types d'obstacles : un cercle ou un segment. L'algorithme de calcul de force appelé *force de collision* fonctionne comme celui de la force de séparation en remplace $\vec{d}$ par le vecteur distance entre l'agent et l'obstacle (le point de l'obstacle le plus proche de l'agent). Grâce aux obstacles segments appelés *Wall*, on peut dessiner des obstacles sous n'importe quelle forme. Des exemples sont donnés à la Figure 2.

2.5 Le "grid"

2.5.1 Présentation du "grid"

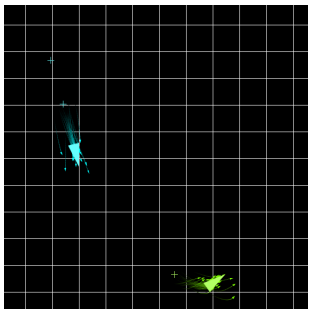


FIGURE 3 – Fenêtre de simulation divisée en une grille

Nous avons choisi de quadriller l'espace en ajoutant une grille représentée sur l'image ci-dessus. Le constructeur de la grille prend en arguments sa résolution verticale et horizontale, il est ainsi possible de faire varier ces paramètres pour influer sur le nombre de cases (appelées mailles, ou Mesh dans notre code). Cette grille n'est pas affichée, c'est un outil qui va nous permettre de diminuer la complexité de calcul pour les algorithmes de *clustering* et de prédiction de la trajectoire.

```
class Grid{
    int ResolutionX; #nombre de lignes
    intResolutionY; #nombre de colonnes
    Mesh[][] boxes; #matrice de Mesh
}
```

La grille est surtout utile car elle possède une matrice de "Mesh" : boxes. Chaque Mesh possède en argument ses coordonnées spatiales (les deux entiers i et j tel que la Mesh en question soit boxes[i][j]), la liste des agents présents dans la maille et la liste de tous les obstacles dans la maille. Chaque fois qu'un agent se déplace, il recalcule sa maille par division entière. S'il change de maille, il s'efface de sa maille précédente (de la liste des agents plus précisément) et se rajoute dans la liste de sa nouvelle maille. Lorsqu'un obstacle est tracé, il s'inscrit dans la liste d'obstacles des mailles qu'il traverse.

```
class Mesh{
    Grid grid;
    int i;
    int j;
    ArrayList<Boid> boids; #toutes les entites presentes dans la maille
    ArrayList<Obstacle> obstacles; #tous les obstacles dans la maille
}
```

La liste d'obstacles de chaque maille sera utilisée pour le calcul de trajectoire et nous allons tout de suite développer l'utilité de la liste d'entités dans une maille.

2.5.2 Calcul de voisins par utilisation de la grille

Afin de repérer les essaims d'entités, chaque individu est obligé de calculer la distance qui le sépare de chaque individu et de la comparer à une distance maximale, en dessous de laquelle deux entités sont considérées comme faisant partie du même essaim. Nous appellerons cette distance le rayon d'observation. Initialement, pour connaître les essaims, chaque entité devait se comparer à toutes les entités de la fenêtre et les ajouter ou non à leur essaim (soit une complexité en n^2, dans le cas où n est le nombre d'entité). Cependant, à présent nous possédons une grille, dont nous supposons que chaque maille (carrée) a un côté de longueur au moins égal au rayon d'observation. Dès lors, il suffit à un agent qui souhaite connaître ses voisins de questionner la maille sur laquelle il se trouve à ce moment. Cette dernière va interroger toutes ses mailles voisines, qui sont au nombre de neuf (celle sur laquelle se trouve l'agent et les huit qui l'entourent) et leur demander de renvoyer les agents qui se situent sur elles. Enfin, l'agent va déterminer sa distance aux agents renvoyés par les mailles, et ne conserver que celles étant à une distance inférieure à la valeur du rayon d'observation.

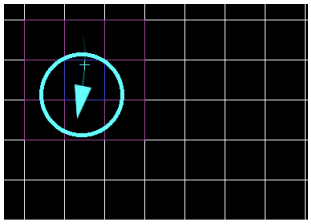


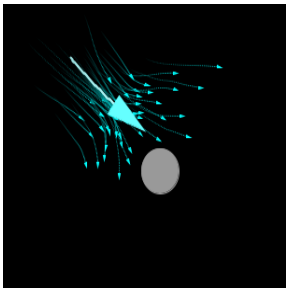
FIGURE 4 – Recherche de voisins

Si l'on prend l'exemple de ce schéma, les voisins de l'entité bleue claire ne peuvent être que dans le cercle bleu clair, soit dans les 9 mailles (violettes et bleues) autour de l'entité. Cette opération réduit considérablement la complexité meilleur cas de la recherche de voisins (qui passe de n^2 à n). Ainsi, maintenant que l'on possède une liste de nos voisins, on va pouvoir détecter les groupes.

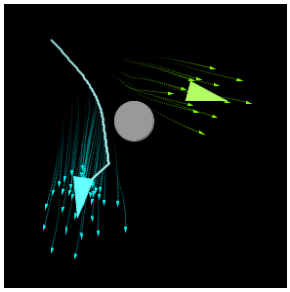
3 Détection de groupes

3.1 Algorithme de *clustering*

L'algorithme que nous utilisons afin de déterminer les différents groupes est un algorithme de détermination des composantes connexes d'un graphe non orienté. Les sommets du graphe sont les agents et deux sommets sont reliés par une arête si et seulement si au moins un des deux est présent dans la liste des voisins de l'autre, déterminée précédemment. Les composantes connexes contenant au moins deux agents sont les groupes, ou *clusters*. Les autres ne sont pas retenues, et les agents correspondant sont considérés isolés (ils seront affichés en rouge). On obtient ainsi une liste de groupe numérotés dans l'ordre dans lequel ils ont été trouvés. Cet algorithme est implémenté dans la méthode *find()* de la classe *Clustering*, le graphe est représenté par une liste d'adjacence, qui est la liste de la liste des voisins de chaque agent.



(a) Avant la séparation



(b) Après la séparation

FIGURE 5 – Détection de la séparation d'un groupe

3.2 Cohérence des groupes après un nouveau calcul

L'algorithme de *clustering* est exécuté à intervalles réguliers, et l'on obtient à chaque fois une nouvelle liste de groupes, potentiellement différentes de celle obtenue au précédent calcul. Nous souhaitons conserver une cohérence entre ces deux listes, pour cela, on attribue un identifiant unique (un entier positif ou nul) à chaque groupe du précédent calcul. Puis, pour chaque nouveau groupe calculé, on compte le nombre d'agents provenant de chaque ancien groupe, et le nombre de ceux qui étaient isolés. Nous en déduisons pour chaque nouveau groupe le cas qui lui correspond parmi les suivants.

3.2.1 L'apparition de nouveaux groupes

Lorsque la majorité des agents du nouveau groupe n'étaient pas assignés à un groupe lors du précédent calcul, on considère qu'il y a création d'un nouveau groupe, on lui attribue un nouvel identifiant non utilisé.

3.2.2 La séparation d'un groupe, ou *split*

Lorsque dans plusieurs nouveaux groupes différents, la majorité des agents de chacun provient d'un même ancien groupe, on considère que l'ancien groupe s'est séparé.

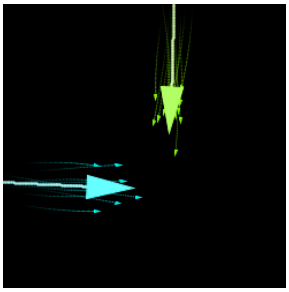
On cherche le groupe le plus nombreux parmi ceux-ci, et on lui attribue l'identifiant de l'ancien groupe. On attribue aux autres de nouveaux identifiants non utilisés. On peut voir un exemple de séparation de groupe à la figure 5.

3.2.3 La fusion de groupes, ou *merge*

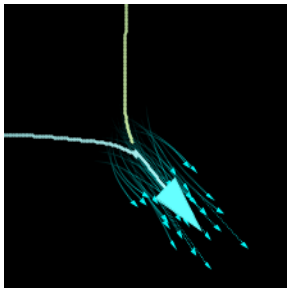
Pour tous les nouveaux groupes qui n'ont été identifiés ni comme des nouveaux groupes ni comme des groupes s'étant séparés, on considère que ce sont des groupes qui existaient déjà avant mais qui n'ont subi que des variations mineures (peu d'agents en plus ou en moins), on leur attribue l'identifiant majoritaire parmi ceux des anciens groupes de ses agents. Ceci nous permet de gérer la fusion des groupes, en effet, lorsque plusieurs groupes se rencontrent, tous les agents qui les composaient se retrouvent dans le même nouveau groupe, qui prend alors l'identifiant le plus représenté. Les autres anciens groupes (minoritaires dans le nouveau) sont désormais vides. On peut voir un exemple de séparation de groupe à la figure 6.

3.3 Calcul du centre des groupes

Après avoir déterminé les différents collectifs d'agents, il est désormais possible de déterminer algorithmiquement le centre de chacun d'entre eux. En effet, pour ce faire, il suffit de faire la somme de toutes les positions des agents du groupe coordonnée par coordonnée, puis de diviser chaque quantité obtenue par le nombre d'agents présents au sein du groupe. Chaque agent étant représenté par un triangle, nous avons décidé de représenter le centre d'un groupe par un triangle de taille plus importante, car il permet de représenter le groupe dans son ensemble si l'on se place à un niveau d'abstraction plus important. Nous avons donc écrit une fonction permettant de calculer ce centre, une autre fonction qui sert à mettre le centre de chaque groupe à jour à chaque itération de la simulation, et une dernière



(a) Avant la fusion



(b) Après la fusion

FIGURE 6 – Détection de la fusion de deux groupes

fonction qui permet de marquer à intervalles de temps réguliers la position de chaque groupe dans l'espace où les agents évoluent, ce qui laisse une trace de la trajectoire empruntée par les groupes.

input : $Voisins$
output: $\vec{position}$
$count \leftarrow 0$
$\vec{position} \leftarrow \vec{0}$
for a *in* $Voisins$ **do**
  $\vec{position} \leftarrow \vec{pos}_a$ // $\vec{pos}_a$ est le vecteur position de a
  $count \leftarrow count + 1$
end
if $count > 0$ **then**
  $\vec{position} \leftarrow \frac{\vec{position}}{count}$
end

Algorithme 5: Algorithme de calcul du centre

4 Prédiction de trajectoire

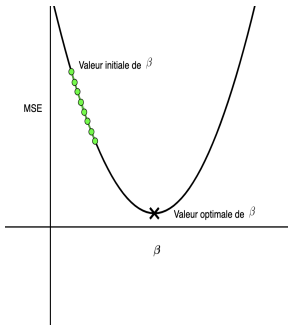
La prédiction de la trajectoire des *clusters* d'agents se base sur celle de leur centre, en utilisant les positions passées de ces derniers et en calculant, grâce à différentes méthodes de régression détaillées ci-après, les futures positions du centre des groupes.

4.1 Régression polynomiale

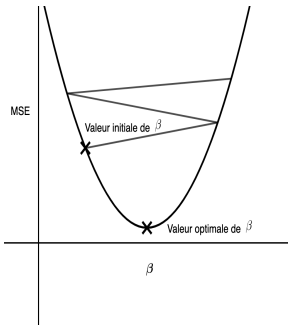
La première régression que nous avons implémenté est la régression polynomiale. Nous utilisons uniquement des régressions linéaires, quadratiques et cubiques mais notre algorithme nous permet d'obtenir une régression polynomiale plus générale de tout degré n ;

4.1.1 Algorithme du gradient

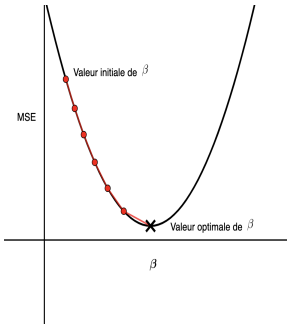
Le tout premier algorithme que nous avons utilisé a été un algorithme de régression linéaire par algorithme de gradient, basé sur ceux proposés par [2] et [3]. Cependant, par la nature aléatoire et courbée des mouvements produits par les groupes d'agents, nous avons décidé qu'une régression polynomiale de plus grand degré nous permettrait d'obtenir un prédiction de trajectoire plus fidèle. Grâce à notre premier algorithme de dimension 1, nous avons pu facilement l'étendre à un algorithme généralisé, capable de produire un régression linéaire d'ordre quelconque. Cet algorithme est décrit dans l'algorithme 6. Supposons qu'on veuille obtenir un régression polynomiale de degré p. On souhaite donc trouver une fonction de la forme : $P(x) = \sum_{k=0}^{p} a_k x^k$. On définit donc dans 1 la fonction du coût, ou *Mean Squared Error (MSE)* qui donne l'écart par rapport à notre fonction actuelle et les points $(y_i)_{i\in[1,n]}$ avec n le nombre de points.



(a) Taux d'apprentissage trop petit



(b) Taux d'apprentissage trop grand (*Overshoot*)



(c) Taux d'apprentissage correct

FIGURE 7 – Algorithme de gradient pour différentes grandeurs de α

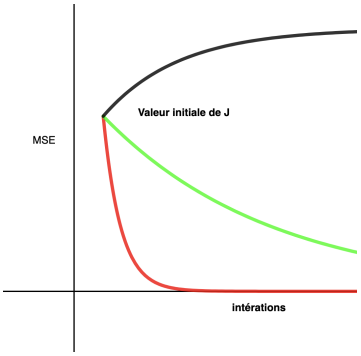


FIGURE 8 – Convergence en fonction de α
Vert : α trop petit Rouge : α correct Noir : α trop grand

$$MSE = J = \frac{1}{n}\sum_{i=1}^{n}(y_i - P(x_i))^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \sum_{k=0}^{p} a_k x_i^k)^2 \tag{1}$$

$$r^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \sum_{k=0}^{p} a_k x_i^k)^2}{\sum_{i=1}^{n}(y_i - \overline{y_i})^2} = 1 - \frac{\sum_{i=1}^{n}(y_i - P(x_i))^2}{var(y)} \tag{2}$$

L'objectif est donc de minimiser cette fonction afin d'obtenir la plus petite erreur possible, c'est-à-dire de rapprocher le plus de 1 le coefficient de détermination r^2, présenté en 2. Pour minimiser cette fonction on utilise la méthode du gradient. Le principe est de soustraire à la *MSE* son gradient (3, 4) afin de le faire décroître vers son minimum. Pour chaque coefficient du polynôme, on effectue donc l'opération suivante :

$$\frac{\partial J}{\partial a_k} = \frac{2}{n}\sum_{i=1}^{n}(y_i - P(x_i))x_i^k = \frac{2}{n}\sum_{i=1}^{n}(y_i - \sum_{j=0}^{p} a_j x_i^j)x_i^k \tag{3}$$

$$a_k = a_k - \alpha\frac{\partial J}{\partial a_k} = a_k - \alpha\frac{2}{n}\sum_{i=1}^{n}(y_i - \sum_{j=0}^{p} a_j x_i^j)x_i^k \tag{4}$$

avec α un coefficient qu'on choisit appelé *Learning Rate*, le taux d'apprentissage. α désigne le pas avec lequel on avance vers le minimum de J. Ce coefficient doit être suffisamment grand pour y arriver avant le nombre d'itération maximale qu'on définit en amont et suffisamment petit pour éviter le *Overshoot*. Dans le cas du *Overshoot*, le coefficient α qu'on "dépasse" le coefficient. Cette situation est représentée à la figure 7. On note $\beta = (a_0, a_1, \ldots, a_{n-1}, a_n)$. La convergence et divergence des différents cas de taux d'apprentissages sont montrées à la figure 8.

input : X : coordonnées x des points, Y coordonnées y des points, p : degré du polynôme, n_{iter} : nombre maximale d'itération, ${r^2}_{min}$: coefficient de détermination minimal, α : coefficient d'apprentissage
output: P : polynôme approximé

$\vec{\beta} \leftarrow init()$ // On initialise $\vec{\beta}$
$var \leftarrow var(Y)$ // Variance de Y
$r^2 \leftarrow 0$ // coefficient de détermination
while $r^2 < {r^2}_{min} k < n_{iter}$ **do**
  for $i = 1$ **to** n **do**
    $\frac{\partial J}{\partial a_k} \leftarrow 0$ // On initialise la dérivée partielle de MSE à 0
    for $k = 0$ **to** p **do**
      $\frac{\partial J}{\partial a_k} \leftarrow \frac{\partial J}{\partial a_k} + \frac{2}{n}(y_i - P(x_i))x_i^k$
    end
  end
  for $k = 0$ **to** p **do**
    $a_k \leftarrow a_k - \alpha * \frac{\partial J}{\partial a_k}$
  end
  $P \leftarrow \sum_{k=0}^{p} a_k X^k$ // On prend le polynôme de coefficients dans β
  $RSS \leftarrow 0$ // On initialise RSS à 0
  for $i = 1$ **to** n **do**
    $RSS \leftarrow rss + (y_i - P(x_i)^2$
  end
  $r^2 \leftarrow 1 - \frac{RSS}{var}$
  $k \leftarrow k + 1$
end

Algorithme 6: Régression polynomiale par méthode du gradient

4.1.2 Algorithme matriciel

La deuxième méthode pour obtenir la régression polynomiale utilise des matrices plutôt qu'une fonction de coût. Dans cette méthode, on impose que $p < n$ et nous utilisons la matrice de Vandermonde [4] [5] construite avec nos valeurs $(x_i)_{i\in[1,n]}$ que nous inversons [6]. Nous obtenons ensuite les coefficients de notre polynôme en multipliant cette matrice par la matrice colonne contenant nos valeurs $(y_i)_{i\in[1,n]}$. La seule contrainte est que la taille de la matrice de Vandermonde soit égale à la taille de notre matrice colonne de coefficients, ce qui impose un coefficient de même degré que le nombre de points. Cette méthode est expliquée dans l'algorithme 7 qui considère le cas $p = n - 1$.

On définit ϵ_i comme l'erreur entre y_i et $P(x_i)$, on obtient la relation 5.

$$\forall i \in [1, n], y_i = a_0 + a_1 x_i + \cdots + a_{p-1} x_i^{p-1} + a_p x_i^p + \epsilon_i \tag{5}$$

On peut donc écrire cette équation sous la forme matricielle 6. La matrice X ci-dessous est celle de Vandermonde.

$$Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^p \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_p \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{pmatrix} = X\beta + \epsilon \tag{6}$$

L'objectif de l'algorithme est de réduire l'erreur (*Squared Vertical Distances*) donnée dans 7 pour obtenir un approximation de β 8. Minimiser RSS revient à approcher le coefficient de détermination R^2 vers 1.

$$RSS(\beta) = \sum_{i=1}^{n} (y_i - \sum_{k=0}^{p} a_k x_i^k)^2 = (Y - X\beta)^{\perp}(Y - X\beta) \tag{7}$$

$$\hat{\beta} = arg \min_{\beta \in R^{p+1}} RSS(\beta) \tag{8}$$

La forme matricielle de β devient donc 9.

$$\hat{\beta} = (X^{\perp}X)^{-1}X^{\perp}Y \tag{9}$$

Dans le cas ou $p = n - 1$, X est une matrice carré, donc grâce à la méthode de calcul de son inverse grâce à l'équation 11 avec la méthode donnée par [6], nous donnons une approximation pour $\vec{\beta}$ dans 10. Les matrices $U^{-1} = (u_{i,j})_{i,j\in[1,n]^2}$ et $L^{-1} = (l_{ij})_{ij\in[1,n]^2}$ sont donnés dans 12 et 13.

$$\hat{\beta} = X^{-1}Y \tag{10}$$

$$X^{-1} = U^{-1}L^{-1} \tag{11}$$

$$u_{ij} = \begin{cases} 0 & si\ i < j \\ 1 & si\ i = j = 1 \\ \prod k = 1,\ k^i \frac{1}{x_j - x_k} & sinon \end{cases} \tag{12}$$

$$l_{ij} = \begin{cases} 0 & si\ j = 1 \\ 1 & si\ i = j \\ u_{i-1,j-1} - u_{i,j-1}x_{j-1} & sinon \end{cases} \tag{13}$$

input : X, Y
output: P : polynôme approximé

1 $X_v \leftarrow Vandermonde(X)$ `// On crée la matrice de Vandermonde à partir de` X
2 $X^{-1} \leftarrow Inverse(X_v)$ `// On inverse` X^{-1}
3 $\beta \leftarrow X^{-1}Y$
4 $P \leftarrow \sum_{k=0}^{p} a_k X^k$ `// On prend le polynôme de coefficients dans` β

Algorithme 7: Régression polynomiale par méthode du gradient

4.2 Recherche du plus court chemin dans un graphe

La dernière méthode de prédiction de chemin utilise la grille que nous avons créé. Pour un essaim donné, nous connaissons sont centre de gravité et donc dans quelle maille se trouve sont centre. Nous confondons donc l'essaim avec le centre de ce maille. Ensuite nous construisons un graphe donc les noeuds sont les mailles. Le poids de chaque arête dépendra de plusieurs paramètres représentera la difficulté que l'essaim passe de la maille de départ de l'arête à la maille d'arrivée de l'arête. C'est-à-dire qu'entre deux mailles, plus son poids est faible, plus l'essaim est probable de passer d'une maille à l'autre. Cette fonction qui associe à chaque arête un poids dépendra de la direction de l'essaim, des obstacles dans la maille d'arrivée et des autres essaims présents dans la maille d'arrivée. Un exemple de ce graphe est donné à la figure 9. Le calcul du poids entre une maille de position $M_{x,y}$ et une autre de position$M'_{x',y'}$ est donné par l'équation 14 :

- θ est l'angle entre le vecteur vitesse de l'essaim et le vecteur entre les deux centres des mailles. La fonction du poids diminue plus l'essaim se dirige dans le sens opposé du vecteur reliant M et M' et croit plus l'essaim se dirige dans le même sens.
- *Obstacles* est l'ensemble des obstacles présents dans la maille M'.
- l_i est la longueur de l'obstacle présent dans la maille. La fonction du poids augmente donc plus il y a de longueur d'obstacles dans la maille d'arrivée.
- α est un coefficient. Dans notre simulation nous prenons $\alpha = 4$.
- d est la longueur de la diagonale d'une maille
- La fonction f renvoie 1 si M' contient un autre essaim et $1 + \alpha$ sinon.

$$p(M, M') = \sum_{o_i \in Obstacles} \frac{l_i}{d}(1 + \alpha \sin^2 \frac{\theta}{2}) f(M') \tag{14}$$

Une fois le graphe obtenu, nous pouvons procéder de plusieurs manières :

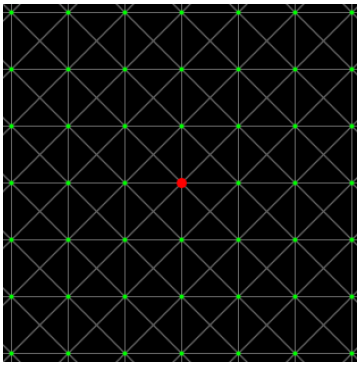


FIGURE 9 – Le graphe obtenu à partir de la grille
Vert : Mailles accessibles Rouge : Maille contenant l'essaim

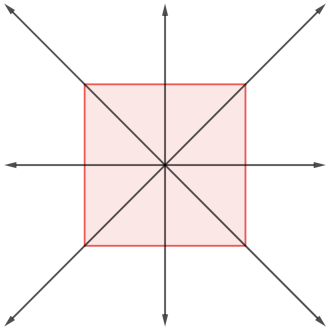


FIGURE 10 – Directions possibles dans le graphe

— Méthode 1 : Nous ne regardons que les voisins de la maille où se trouve l'essaim et en prenant l'arête de poids minimal, on prédit ou sera l'essaim dans un temps $t \simeq \frac{d}{2\|\vec{v}\|}$ avec d la longueur d'une arête de la maille et $\vec{v}$ la vitesse de l'essaim.
— Méthode 2 : Nous pouvons également prendre un graphe plus grand (un carré de taille $> 2n$ dont le centre est notre maille contenant l'essaim car un déplacement de n mailles depuis le centre en sortira pas du graphe) où l'on effectue la procédure précédente. C'est-à-dire qu'à partir de la maille de l'essaim on trouve la prochaine maille en suivant l'arête de poids minimale dans les voisins. On refait cette opération sur la maille obtenue jusqu'à s'être déplacer de n mailles. La maille sur laquelle nous finissons sera donc la maille dans laquelle se trouvera l'essaim dans un temps $t \simeq n\frac{d}{2\|\vec{v}\|}$.
— Méthode 3 : Enfin, en prenant le même graphe que dans la situation précédente, nous appliquons Dijkstra à ce graphe et nous prenons, dans l'arbre résultant, la plus courte branche de taille n. La maille au bout du chemin sera donc la maille dans laquelle se trouvera l'essaim dans un temps $t \simeq n\frac{d}{2\|\vec{v}\|}$. Un exemple est donné par la figure 11.

Dans la suite, nous appellerons ces trois méthodes respectivement G1, G2 et G3.

Avec ces trois méthodes, comme nous choisissons un chemin taille poids faible (dans certains cas, de poids minimal), et comme le poids diminue avec la possibilité que l'essaim passe par cette arête, nous en déduisons donc une prédiction de trajectoire de maille en maille. La précision de chaque méthode sera évalué par la suite. Une application de cette méthode sur un essaim dans un chemin délimité par un *Wall* de chaque côté est montré à la figure 12.

5 Étude de cas d'école

Dans cette partie nous allons étudier quelques situations de simulation et analyser la pertinence de nos méthodes de prédictions de trajectoire. Pour cela nous étudierons l'écart entre la trajectoire réelle et la trajectoire prédite. Pour cela, notons la position réel p_{t_i} à un moment t_i et sa position prédite

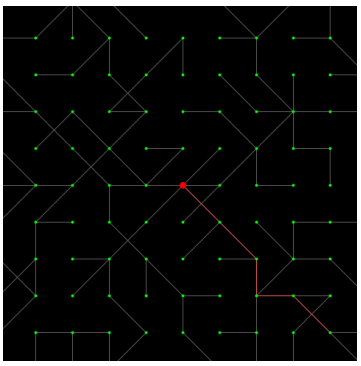
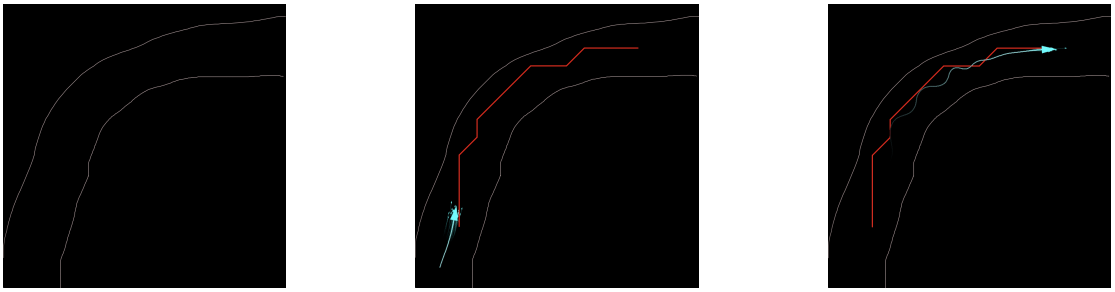


FIGURE 11 – Méthode avec Dijkstra pour $n = 5$



(a) Chemin initial (b) Essaim avec prédiction (c) Trajectoire réelle

FIGURE 12 – Chemin réel avec prédiction G2

à ce même moment $\hat{p}_{t_i}$. Nous évaluerons nos algorithmes par le *MSE* correspondant, c'est-à-dire avec $MSE = \frac{1}{n}\sum_{i=1}^{n} \|p_{t_i} - \hat{p}_{t_i}\|^2$.

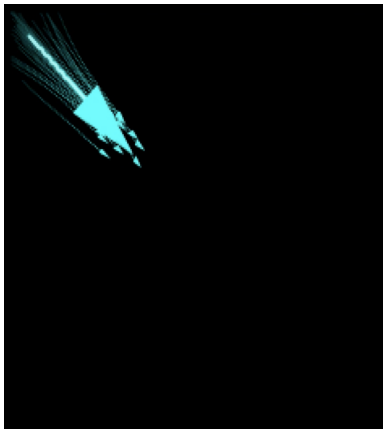
Nous verrons également la pertinence de la simulation en comparant des phénomènes de physiques mécaniques avec les mouvement des essaims.

5.1 Mouvement rectiligne uniforme

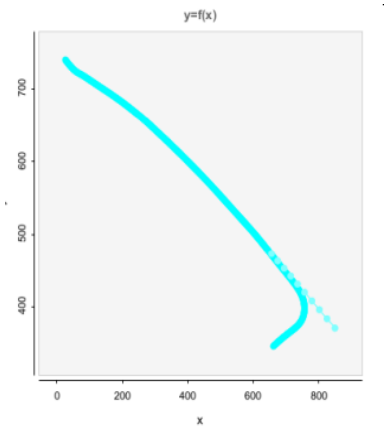
Étudions dans un premier temps une trajectoire rectiligne uniforme d'un essaim 13, c'est-à-dire qu'à part les forces internes de l'essaim (autrement dit les trois forces entre les agents composant l'essaim), il n'y a aucune force. Cette situation est donc sans autres essaims et sans obstacles. Comme le trajet est rectiligne, on s'attend à une régression polynomiale 14 (dans ce cas linéaire) très précis. Cependant, pour la méthode G2 16, comme les directions possibles sont finis (il y en a 8 en tout, voir figure 10), la trajectoire prédite sera celle de direction la plus proche à la direction réelle de l'essaim.

5.1.1 Régression

Comme attendu, la figure 14a nous montre une régression linéaire confondue avec la trajectoire de l'essaim. Pendant le simulation, tant que l'essaim avançait de façon rectiligne, nous avons obtenu un coefficient de détermination r^2 entre 0.92 et 0.99. Cependant, l'essaim est toujours soumis à ses forces internes. Ainsi, il y a toujours quelques variations aléatoires par rapport à un mouvement purement rectiligne. On peut voir à la figure 14b, l'essaim n'est plus sur la même trajectoire prédite par la régression au lancement de la simulation (Il y a un décalage par rapport à la régression initiale, sur la figure les ancienne régression sont plus foncés et les récentes plus claires). Cependant, la régression est effectué à intervalle régulier donc les nouvelles régressions correspondent bien à la trajectoire de l'essaim. Nous observons même l'apparition d'un courbure car l'essaim tourne, ce qui augmente les coefficients de degré > 1 dans le polynôme résultant de la régression.

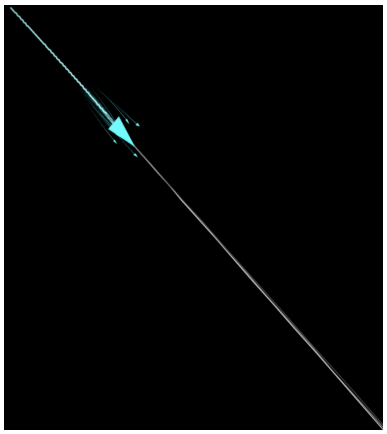


(a) L'essaim au lancement de la simulation

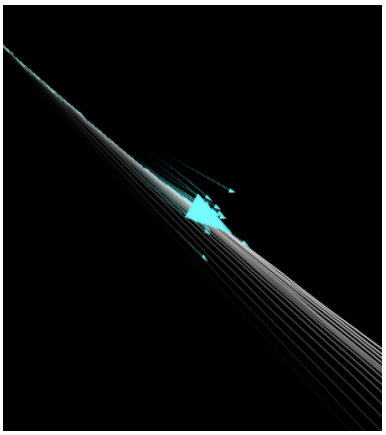


(b) Trajectoire complète

FIGURE 13 – Mouvement rectiligne uniforme

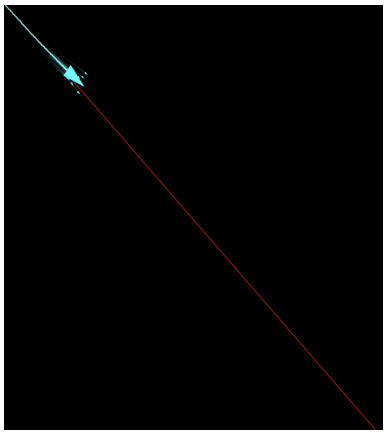


(a) Régression polynomiale au lancement

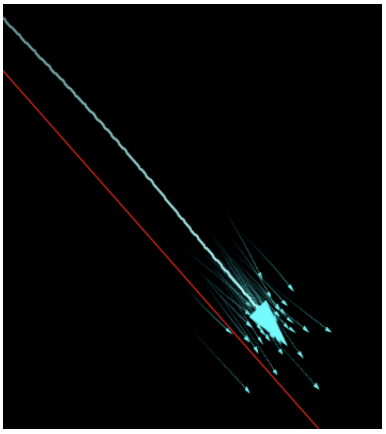


(b) Régression polynomiale pendant la simulation

FIGURE 14 – Algorithme de gradient pour différentes grandeurs de α

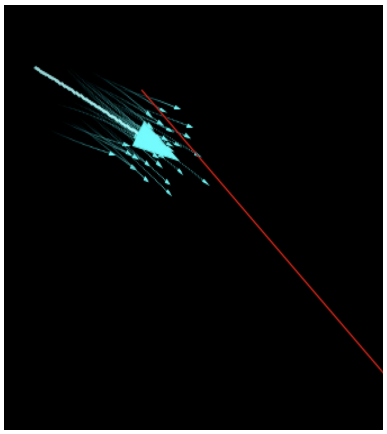


(a) Parcours de graphe au lancement

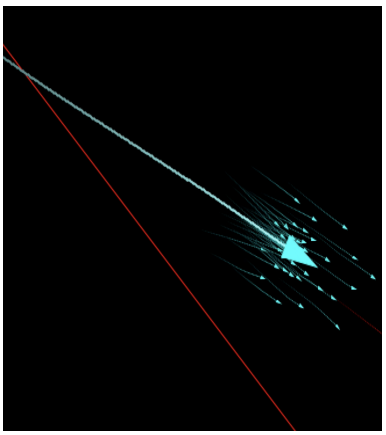


(b) Parcours du graphe après un déplacement

FIGURE 15 – Parcours de graphe pour un angle accessible



(a) Parcours de graphe au lancement



(b) Parcours du graphe après un déplacement

FIGURE 16 – Parcours de graphe pour un angle non accessible

5.1.2 Parcours de graphe

Contrairement à la régression, la méthode G2 n'est que effectuée au lancement de la simulation et nous observons si la trajectoire réelle est proche de cette unique trajectoire prédite au lancement. Nous voyons à la figure 15a lorsque nous initialisons l'essaim avec un angle correspondant à la direction possible du graphe (figure 10). On obtient une prédiction similaire à la régression. Malgré le mouvement bruité de l'essaim, en moyenne l'angle de direction reste constant. Ainsi, à la fin de la simulation, nous pouvons voir à la figure 15b que la trajectoire réelle de l'essaim est décalée par rapport à la trajectoire prédite mais la directions reste plutôt constant.

Dans le cas où nous prenons un angle non accessible par le graphe, le parcours du graphe favorisera les arête dont la direction est la plus proche de la direction de l'essaim, mais on aura un décalage par rapport à sa vraie direction. Ainsi, comme nous pouvons les voir à la figure 21a dès le lancement, la prédiction de trajectoire semble être éloignée d'une trajectoire parallèle à la direction de l'essaim. Cette divergence est marqué de plus en plus avec le temps. La figure 15b montre que pendant le simulation, l'erreur croît linéairement.

5.2 *Merge* de groupes

Nous avons étudié le comportement d'un groupe résultant de la fusion de deux groupes. En particulier, nous souhaitions comparer le vecteur vitesse du groupe au vecteur vitesse correspondant à un modèle physique.

5.2.1 Définition du modèle

Le vecteur théorique est obtenu par la loi de conservation de la quantité de mouvement sur le système de l'ensemble des agents. Cela se traduit par l'équation 15. Où N_1 et N_2 sont les nombres d'agents respectifs des deux groupes avant la fusion, $\vec{v_1}$ et $\vec{v_2}$ leurs vitesses respectives, et enfin N le nombre total d'agent et $\vec{v}$ le vecteur vitesse du groupe résultant de la fusion.

$$N\vec{v} = N_1\vec{v_1} + N_2\vec{v_2} \tag{15}$$

5.2.2 Obtention des résultats simulés

Afin de vérifier ce modèle, nous initialisons deux nuées d'agents, de population respectives N_1 et N_2, se dirigeant l'un vers l'autre avec des directions orthogonales. Ceci est réalisé en appuyant sur la touche 'M' après avoir saisi dans le code les valeurs de N_1 et N_2. Après enregistrement de la simulation, on calcule les vecteurs vitesses $\vec{v_1}$ et $\vec{v_2}$ en effectuant une régression linéaire sur chacun des deux axes en fonction du temps sur les 70 dernières positions des groupes avant la fusion. On calcule le vecteur vitesse $\vec{v}$ de la même façon sur 70 positions du groupe final (après un délai de 20 itérations).

5.2.3 Observation des résultats

Ces résultats sont représentés sur la figure 17, ainsi que le vecteur théorique (en gris) obtenu par la loi de conservation de la quantité de mouvement.

Afin de comparer les résultats simulés et prédis, on définit l'angle relatif du groupe final comme le rapport de sa différence angulaire avec l'angle du vecteur $\vec{v_2}$, sur la distance angulaire entre $\vec{v_1}$ et $\vec{v_2}$. La norme ainsi que cet angle relatif sont représentés sur la figure 18 en fonction du rapport N_1/N. On obtient un écart relatif moyen de 0.178 pour l'angle relatif, et 0.140 pour la norme de la vitesse.

5.2.4 Analyse des résultats

On remarque que la norme lors de la simulation est systématiquement supérieure à celle donnée par le modèle.

5.3 *Split* de groupes

Un autre phénomène que nous avons étudié est le *Split* : la séparation d'un essaim en deux nouveaux essaimes par une obstacle. Ce cas à lieu lorsque qu'un groupe arrive face à un obstacle et que la force de répulsion de l'obstacle dépasse les forces de cohésion et d'alignement. Ainsi, certains agents partent d'un côté de l'obstacle et les autres agents partent de l'autre côté. Cette situation crée donc deux nous groupes d'agents dont la trajectoire à été légèrement déviée par rapport à la trajectoire de l'essaim originale. Ce phénomène est montré à la figure 19

5.3.1 Régression

La figure 19 montre également la régression polynomiale pendant le *split*. On remarque à la figure 19a que la régression de détecte pas l'obstacle et fait comme si l'essaim se diriger de façon rectiligne. En effet, la régression ne prend que en compte les positions passées de l'essaim et ne peut donc pas prédire la variations de trajectoire futures liées à la présence d'obstacles autour de l'essaim.

Dans le figure19b, on voit que le *split* commence, mais comme l'essaim vient juste de changer de direction, les nouveaux points on un poids faibles devant les points de l'ancienne trajectoire rectiligne. Ainsi, la régression nous donne toujours une trajectoire dans la même direction.

Cependant, après le *split*, la régression a suffisamment de points $post - split$ pour pouvoir détécter la séparation. De plus, comme l'essaim à eu une trajectoire courbée, la régression prédit une continuité de cette courbure à la figure 19c.

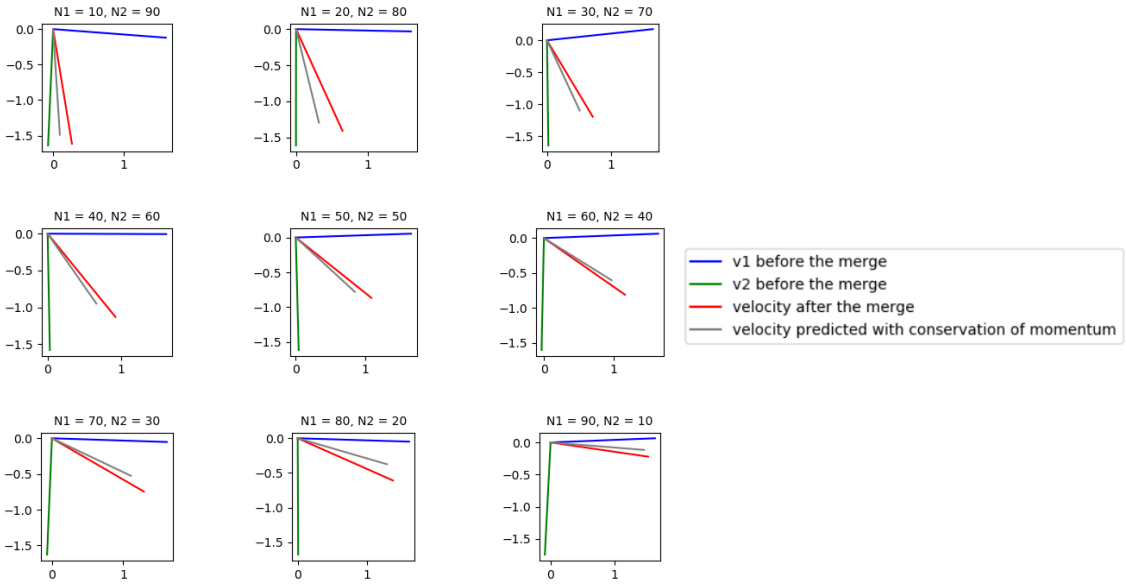


FIGURE 17 – Résultats simulés et prédits

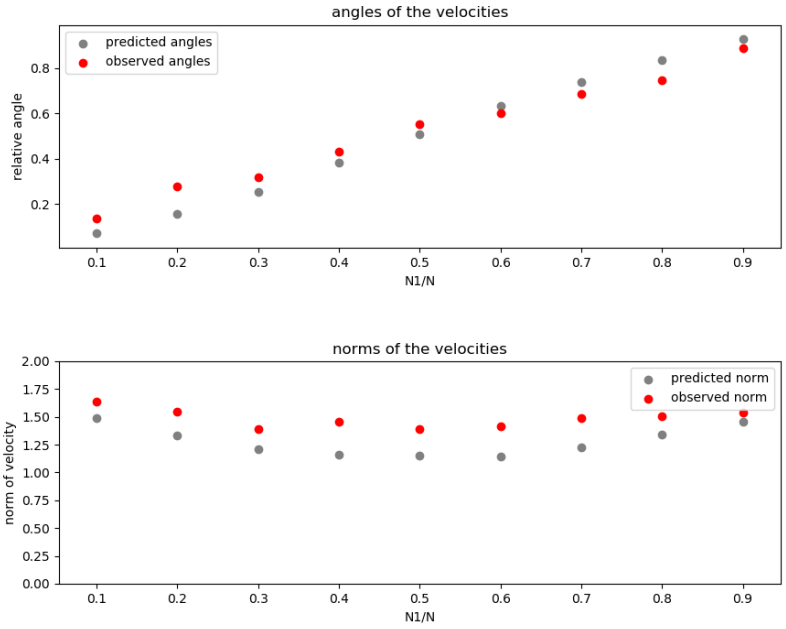
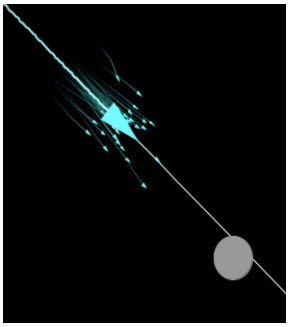
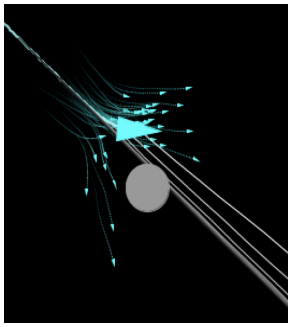


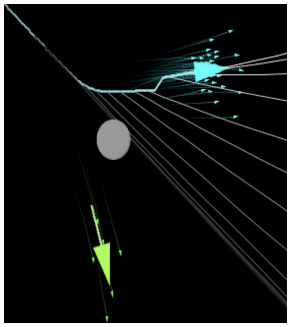
FIGURE 18 – Angles et normes des vitesses de groupe



(a) Essaim avant un *split*



(b) Essaim pendant un *split*



(c) Deux essaims après un *split*

FIGURE 19 – *Split* d'un essaim avec régression polynomiale

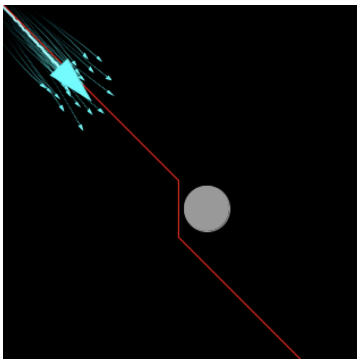


FIGURE 20 – Angles et normes des vitesses de groupe

5.3.2 Parcours de graphe

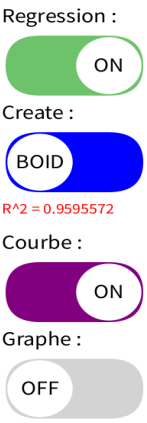
La prédiction de la trajectoire pendant et après le *split* avec la méthode G2, montrée dans la figure 20 montre que cette méthode, contrairement à la régression polynomiale, prend bien en compte la présence des obstacle et peut donc prédire une déviation par rapport à cet obstacle. Cependant, comme cette algorithme n'a que accès à la direction actuelle de l'essaim, il fait comme si à tout moment, la direction est la même que dans l'état initiale. Ainsi, après la déviation par rapport à l'obstacle que l'algorithme prédit, la direction de l'essaim serait la même qu'au début, et non une direction qui s'éloigne de l'obstacle. Ainsi, il y a une erreur de prédiction de trajectoire en sortie du *split*. Il est également important de note que cette prédiction n'est que valable pour un des deux essaimes sortants.

6 Manuel d'utilisation

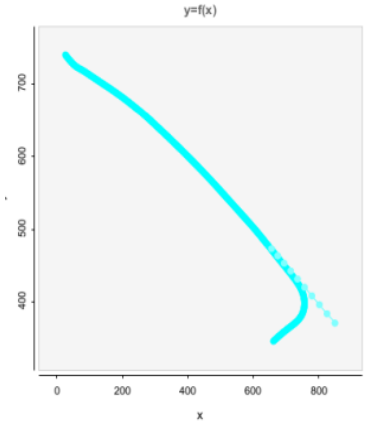
Téléchargez Processing via ce lien : https://processing.org/download. Importer les deux bibliothèques suivantes : Matrix Pallav et Grafica. Dans l'environnement Processing, cliquez sur "*Fichier*", puis sur "*Ouvrir*", et ouvrez n'importe quel fichier du dossier contenant le code. Exécutez le programme en cliquant sur le triangle bleu en haut à gauche de la fenêtre de Processing. À ce stade, le bouton Create vous permet de choisir ce que vous allez dessiner.

Vous pouvez rester dans le mode Boid (mode par défaut), auquel cas vous pouvez créer un essaim d'agents en cliquant sur un point de l'espace de simulation et en déplaçant la souris tout en maintenant le clic enfoncé, puis en relâchant vous choisissez la direction des agents. Vous pouvez également passer au mode Obstacles en cliquant sur le bouton Create, et alors vous pouvez dessiner la forme d'obstacles que vous souhaitez en maintenant le clic enfoncé dans la fenêtre de simulation pour créer l'obstacle. Vous pouvez tout à fait revenir au mode Boid pour créer à nouveau d'autres essaims d'agents.

Vous pouvez également décider d'afficher la prédiction de la trajectoire des groupes basée sur la régression, en cliquant sur le bouton Regression. Vous pouvez ensuite décider de désactiver cet affichage en cliquant à nouveau sur ce même bouton. Enfin, vous pouvez afficher la trajectoire des groupes dans



(a) Les différents boutons de l'application



(b) Régression polynomiale (pointillés en bleu très clair) et chemin effectivement emprunté (bleu clair)

un graphe en cliquant sur le bouton Courbe, et le graphe s'affichera, avec une courbe de couleur différente pour chaque groupe.

Vous pouvez également, en vous assurant d'avoir cliqué à l'intérieur de l'espace de simulation, utiliser certaines touches du clavier pour déclencher des événements prédéfinis. Avec la touche 'M', vous déclenchez l'apparition de deux essaims d'agents qui vont fusionner en un seul groupe. En cliquant sur la touche 'P', vous déclenchez l'apparition d'un obstacle circulaire et d'un essaim d'agent qui va venir rencontrer cet obstacle, ce qui va engendrer une séparation de ce groupe en deux groupes distincts.

Il est également possible d'enregistrer la trajectoire des essaims, vous aurez pour cela besoin de télécharger Python et la bibliothèque Matplotlib (par exemple grâce à la commande *pip install* dans la console Python). Ensuite, lorsque vous exécuterez la simulation sous Processing, vous pourrez, au moment voulu, cliquer sur la touche S du clavier pour enregistrer la trajectoire des groupes d'agents du début de la simulation jusqu'à l'instant courant. Un fichier csv dont le nom est au format "*deplacement_JJ_MM_HH_mm_ss.csv*", selon le jour (JJ), le mois (MM), l'heure (HH), les minutes (mm) et les secondes (ss) où l'enregistrement a été effectué, sera créé dans le sous-dossier "*csv*" du dossier "*Swarm_Simulation*". Pour visualiser les courbes, sortez du dossier "*Swarm_Simulation*" et cliquez sur le dossier "*tracer un graphe*", puis sur le dossier "*graph*", et ouvrez le fichier Python "*fonction en Python.py*". Avant de l'exécuter, modifiez la valeur de la variable *chemin* en la remplaçant par l'emplacement du fichier csv enregistré sur votre machine dans le dossier "*csv*". Vous pouvez maintenant exécuter le script Python, et les courbes apparaîtront sur votre écran. Elles seront au nombre de quatre : une première en 3D, qui trace les coordonnées x et y des groupes en fonction du temps, une deuxième qui trace la coordonnée x des groupes en fonction du temps, une troisième qui trace la coordonnée y des groupes en fonction du temps, et une quatrième qui trace la coordonnée y en fonction de la coordonnée x. Ces courbes permettent de voir si des fusions (*merge*) ou des séparations (*split*) de groupes ont eu lieu, ou si au contraire un seul groupe a subsisté durant la simulation. Elles permettent également d'apprécier la forme des trajectoires prises par les groupes, et leur interaction avec leur environnement, que ce soit avec les murs entourant l'espace de simulation ou que ce soit avec divers obstacles précédemment ajoutés selon votre bon vouloir.

Par ailleurs, le bouton Courbe permet d'afficher un graphe contenant la trajectoire des groupes d'agents actualisée en temps réel, chaque courbe étant de la couleur de son groupe. Il est possible de masquer cet affichage en cliquant à nouveau sur ce même bouton. Nous pouvons également visualiser l'écart entre prédiction de trajectoire par régression polynomiale et chemin effectivement emprunté. Pour ce faire il faut tracer un groupe d'agents, activer la régression et afficher la courbe ; il faut ensuite se placer dans la fenêtre qui affiche les courbes en s'assurant de cliquer dans cette dernière et maintenir la touche 'K' enfoncée. La prédiction apparaît alors d'une couleur plus claire que celle du groupe, en pointillés. (Attention, cette fonctionnalité ne marche qu'avec un seul groupe pour l'instant).

Enfin, le dernier bouton Graphe permet d'afficher la prédiction de la trajectoire par parcours de graphe, seulement son affichage ne peut se faire qu'une seule fois, elle ne peut pas être actualisée.

7 Conclusion

Au cours de ce projet sur le suivi des mouvements collectifs, nous nous sommes rendus compte de la complexité et du caractère vaste et passionnant de ce sujet. En effet, nous avons pu laisser libre cours à notre créativité, dans le sens où aucune approche ne nous a été imposée, et grâce au fait que nous avons mené notre travail de façon autonome, bien que l'encadrement nous ait permis de ne pas nous perdre dans des itinéraires qui auraient abouti à des impasses grâce aux nombreux échanges avec notre encadrante et ses précieux conseils tout au long de l'avancée de notre projet. Par ailleurs, le fait d'être parvenus à obtenir des résultats qui s'apprécient visuellement nous a beaucoup satisfaits, car parfois il est malaisé de se rendre compte de l'étendue du travail fourni. La partie graphique de ce projet étant prédominante, cette déconvenue n'a pas eu lieu car nous avons constaté les progrès du rendu de notre code en temps réel, ou quasiment. Le fait de travailler en groupe nous a permis de se répartir les tâches, de discuter de l'orientation que nous souhaitions que le projet prenne et des concepts inhérents aux bases de la simulation informatique utilisée jusqu'aux détails de celle-ci, ainsi que de nous entraider dans la réalisation de tâches diverses allant de l'écriture d'une nouvelle classe qui permettra améliorer le projet, au test du bon fonctionnement d'une nouvelle fonctionnalité en passant par le débogage d'une fonction qui comporte une erreur difficile à débusquer. De plus, nous pouvons souligner un dernier élément, et non des moindres, qui est l'assiduité systématique de chacun des membres du groupe, qui a permis d'établir un climat de confiance et une bonne ambiance de travail au sein du collectif. Ceci a eu pour effet bénéfique l'épanouissement de chacun des membres participant à notre projet, et a grandement contribué au fait que nous ayions apprécié travailler sur ce projet durant les deux semaines et que nous ayions bien profité de cette opportunité qu'est le PAF en découvrant un domaine qui nous était inconnu, sur lequel nous avons beaucoup appris, et qui peut encore largement nous surprendre par sa richesse.

Références

[1] Daniel Shiffman. *The Nature of Code : Simulating Natural Systems with Processing.* 2012.

[2] Suvarna Galawi. Linear regression in machine learning. *Data Science Blogathon*, 2021.

[3] Mohit Gupta. Gradient descent in linear regression. *GeeksforGeeks*, 2021.

[4] Polynomial regression, wikipedia.

[5] Eduardo Garcia-Portugues. *A Short Course on Nonparametric Curve Estimation.*

[6] L. Richard Turner. Inverse of the vandermonde matrix with applications. *NASA*, 1966.