INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Projet de Physique P6

STPI/P6/2024 – 27

D'UN VOL D'ETOURNEAUX AU BANC DE SARDINES : PEUT-ON MODELISER LEUR COMPORTEMENT ?

Etudiants :

Lila BERTRAND-ADAM **Lilou BONCHE**

Victor DUBUC **Raphael FOURNEAUX**

Charlotte HILLAIRET **Elsa VARIN**

Enseignant-responsable du projet :

Jérôme YON

Cette page est laissée intentionnellement vierge.

Date de remise du rapport : ***15/06/2024***

Référence du projet : ***STPI/P6/2024 – 027***

Intitulé du projet : ***D'un vol d'étourneaux au banc de sardines : peut-on modéliser leur comportement ?***

Type de projet : ***Modélisation et simulation numérique***

Objectifs du projet *(10 lignes maxi)* :

L'objectif de ce projet est de simuler le comportement d'un banc de sardines (ou d'un vol d'étourneaux) à l'aide d'un programme informatique. Il faut pour cela déterminer des lois régissant le comportement de ces animaux et les transcrire sous forme de code informatique.

Mots-clefs du projet *(4 maxi)* : ***Simulation, poissons, comportement.***

TABLE DES MATIERES

TABLE DES ILLUSTRATIONS

1. INTRODUCTION

Tout au long de notre journée, nous sommes confrontés à des densités de foule très différentes. Du confort de la rue déserte à l'inconfort de la rame de métro en heure de pointe, nous expérimentons toutes sortes de situations qui provoquent en nous des comportements que l'on pourrait penser aléatoires. En effet, les mouvements de foule sont souvent associés au chaos et à la désorganisation. En 2010, à l'occasion de la Love Parade - un festival annuel de musique électronique organisé en Allemagne - un mouvement de foule dans un tunnel a provoqué dix-neuf morts et des centaines de blessés. Mais, si la foule peut se révéler très dangereuse, elle ne peut cependant pas être qualifiée de "désorganisée". En effet, même dans la confusion, elle suit des lois et des règles précises que Mehdi Moussaid, chercheur en science cognitive à l'institut Max Planck de Berlin, a étudiées. Cette étude du comportement des foules - qui se décompose en trois grands domaines d'étude : le déplacement collectif, la contagion sociale et l'intelligence collective - permet d'anticiper les mouvements de foules, d'organiser des plans d'évacuation et ainsi d'éviter que des catastrophes comme celle de Love Parade ne se reproduisent.

Mais le comportement des foules humaines n'est pas le seul à intriguer les scientifiques. En effet, le chercheur Mehdi Moussaid voue une partie de son travail à l'étude du comportement des bancs de poissons et des nuées d'oiseaux. Des sortes de mouvements de foules animales, à la différence qu'aucun individu n'y meurt heurté par ses congénères. Et ce même en présence de groupes très denses de millions d'individus, soumis à des niveaux de stress importants, comme des attaques de prédateurs par exemple. Les animaux ont donc beaucoup à nous apprendre, et les étudier permet des avancées dans de nombreux domaines. Les bancs de poissons nous offrent par exemple des leçons de protection mutuelle et d'efficacité par leur organisation collective élaborée. Ils permettent également de mettre en lumière les interactions sociales chez les animaux et d'ainsi mieux comprendre celles chez les humains. Enfin, l'étude mathématique du mouvement des poissons peut aider à comprendre comment des systèmes complexes peuvent émerger à grande échelle à partir de comportements individuels régis par des règles simples [1][2][3].

Notre projet intitulé *"D'un vol d'étourneaux au banc de sardines : peut-on modéliser leur comportement ?"* a pour objectif de modéliser mathématiquement le comportement individuel des poissons évoluant en banc. Notre rapport s'articule donc en plusieurs parties : après avoir exposé la méthodologie et l'organisation que nous avons adoptées, nous nous pencherons sur le travail réalisé et les résultats obtenus. Nous consacrerons une section à l'exposition des modèles existants dont nous nous sommes inspirés, puis nous rentrerons dans les détails de celui que nous avons développé. Nous consacrerons donc les sections suivantes à l'étude individuelle des notions de répulsion, d'alignement et d'attraction, puis à l'étude du programme complet. Nous conclurons ensuite en proposant des possibilités d'amélioration du programme.

Pour toute la suite du rapport et pour la réalisation du code, nous parlerons uniquement de bancs de poissons pour plus de clarté, mais le comportement des vols d'étourneaux peut être modélisé de manière analogue.

2. METHODOLOGIE ET ORGANISATION DU TRAVAIL

2.1. DESCRIPTION DE L'ORGANISATION ADOPTEE

Nous avons d'abord employé les premières séances du projet à nous familiariser avec le sujet, de part de nombreuses recherches effectuées individuellement et mises en commun lors des séances de projet. Nous avons ainsi découvert tous les modèles comportementaux déjà existants, et nous avons pu piocher dans chacun d'entre eux les notions qui nous paraissaient intéressantes et nécessaires à intégrer dans notre modèle. Les concertations terminées, nous avons pu nous atteler à la réalisation de notre modèle et à son implémentation. Nous avons alors scindé le groupe en deux, en fonction des points forts et des facilités de chacun. Victor et Lila - ayant déjà manipulé Python grâce à leurs filières - se sont naturellement proposés pour retranscrire les idées dans le langage informatique. Cela nous a permis de gagner en efficacité et de laisser le reste du groupe, alors composé de Charlotte, Raphaël, Lilou et Elsa, s'atteler à l'étude et la mise au point des trois phénomènes clés de notre modèle : la répulsion, l'attraction et l'alignement. Là encore, nous avons divisé le sous-groupe : Charlotte et Elsa se sont attelées à l'étude assez longue de la répulsion, et Lilou et Raphaël se sont occupés des deux autres notions d'attraction et d'alignement. Victor et Lila ont alors implémenté au fur et à mesure nos découvertes, en procédant à des rectifications fréquentes pour prendre en compte nos remarques et les remarques du professeur encadrant M. Yon. Une fois le programme terminé, nous nous sommes partagés les différents tests à effectuer entre tous les membres du groupe, afin que chacun puisse manipuler et comprendre le programme. La rédaction du rapport s'est faite tout au long du projet, avec une accentuation vers sa fin lorsque le programme a été finalisé. L'entraide a également été un pilier de notre organisation, et il arrivait souvent que l'on aille au-delà de la tâche qui nous avait été attribuée. Lilou s'est par exemple joint plusieurs fois à Victor et Lila pour apporter ses idées dans l'écriture du programme en Python. De même, Charlotte et Elsa ont apporté leur aide pour l'élaboration de l'analyse descendante du programme. Lila a également activement participé à l'étude de l'attraction et de l'alignement avec Lilou et Raphaël.

2.2. ORGANIGRAMME DES TACHES REALISEES ET DES ETUDIANTS CONCERNES

Figure 1 : Organigramme des tâches réalisées et des étudiants concernés

3. TRAVAIL REALISE ET RESULTATS

3.1. RESULTATS DES RECHERCHES SUR LES MODELES EXISTANTS

3.1.1. LE MODELE DES TROIS CERCLES CONCENTRIQUES

Le comportement collectif des poissons suscite l'intérêt des scientifiques depuis longtemps. Dès les années 1930, des théories sur le mouvement collectif des poissons émergent : ce dernier devrait en fait son organisation exemplaire à la télépathie. En effet, d'après Edmund Selous, il existerait un poisson dirigeant au sein de chaque banc de poissons, qui communiquerait ses déplacements par télépathie à tous ses congénères. De ce mécanisme résulterait les chorégraphies aquatiques si structurées qu'offrent les bancs de poissons. Ce n'est qu'à la fin du XXème siècle que cette théorie se heurte à des modèles plus plausibles, comme celui des trois cercles concentriques. Ce modèle est pour la première fois évoqué par le chercheur japonais Ichiro Aoki en 1982 dans le journal A Simulation Study on the Schooling Mechanism in Fish, NIPPON SUISAN GAKKAISHI, 1982. En parallèle, l'informaticien américain Craig Reynolds décrit également le comportement des poissons par un modèle mathématique similaire en 1987. Finalement, ces deux découvertes analogues sont mises en commun en 2002 par Iain Couzin dans l'article coécrit *Collective memory and spatial sorting in animal groups. Journal of theoretical biology, 2002*.

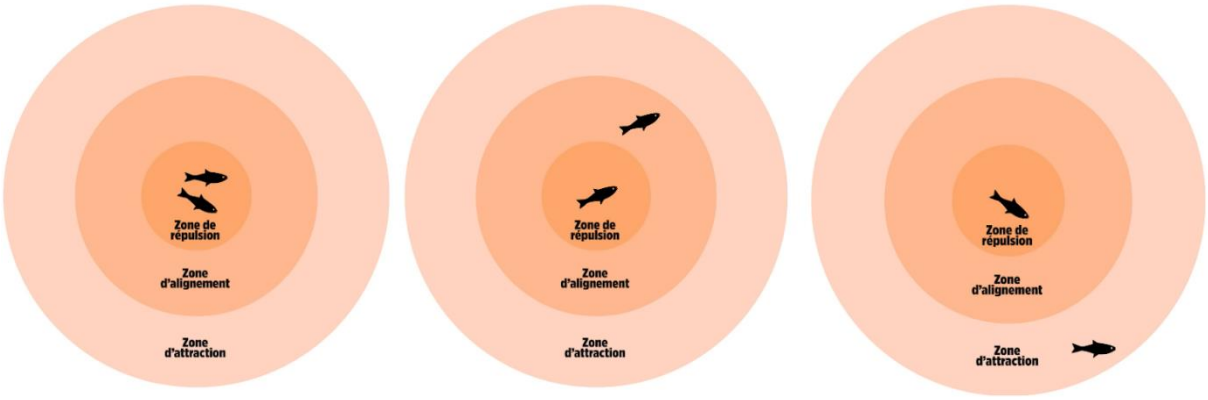
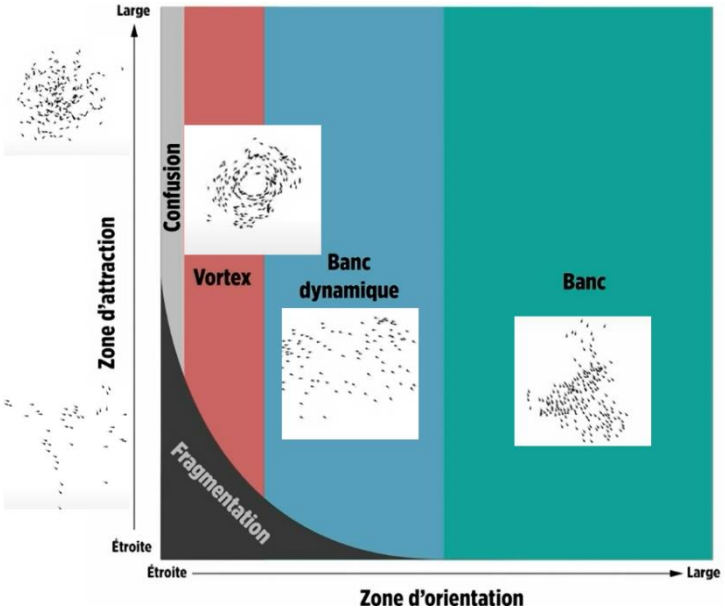


Figure 2 : Schémas des trois situations engendrées par le modèle des trois cercles concentriques (de gauche à droite : illustration des notions de répulsion, d'alignement et d'attraction) [1]

Le modèle des trois cercles concentriques s'appuie sur trois notions essentielles dans les déplacements collectifs : la répulsion, l'alignement et l'attraction. Ces trois principes donnent lieu à la considération de trois cercles concentriques centrés sur chacun des poissons constituant le banc. Le mouvement de ces derniers dépendra donc de la position des poissons voisins. Si un poisson se trouve dans la zone de répulsion du poisson étudié, ce dernier va alors tenter de s'en éloigner pour éviter la collision. En revanche, si un poisson se positionne dans la zone d'alignement du poisson sur lequel l'étude se porte, alors ce dernier cherchera à modifier sa trajectoire afin de s'aligner sur son voisin et de conserver cette distance idéale entre son congénère et lui. Enfin, s'il s'avère qu'un poisson voisin au poisson d'étude se place dans le dernier cercle - la zone d'attraction - ce dernier tentera de diminuer la distance le séparant de son voisin en se rapprochant, dans le but de combler d'éventuels trous dans la formation aquatique. [1]

Plusieurs points de ce modèle sont à relever. La première observation est que l'on ne considère que les poissons se situant dans l'un des trois cercles concentriques, qu'ils soient deux ou quinze. Cela implique que si le poisson est isolé et qu'aucun autre de ses congénères ne se trouve dans un de ses cercles concentriques, son comportement devient imprévisible, du moins avec ce modèle. Il s'agit d'un défaut que l'on tentera de corriger avec notre modèle.

Un autre point important de cette modélisation, plus global, est que l'on ne considère pas le banc de poissons dans son intégralité, mais bien chaque individu constituant ce banc. En effet, ce modèle suggère que l'on parcourt chacun des poissons et qu'on leur applique individuellement les lois de répulsion, d'alignement et d'attraction. Pourtant, on voit bien émerger les chorégraphies aquatiques dont les poissons sont les rois lorsqu'on applique ce modèle à un banc de poissons. Le modèle des trois cercles concentriques illustre donc comment des systèmes complexes peuvent émerger à grande échelle à partir de comportements individuels régis par des règles simples.



Dernier point, concernant les chorégraphies collectives en elles-mêmes, on observe dans ce modèle qu'elles changent de nature en fonction de la valeur des différents rayons des cercles de répulsion, d'alignement et d'attraction. À partir de cette observation, Iain Couzin a établi un diagramme de phase indiquant la formation prise par les poissons en fonction de largeur des zones d'attraction et d'alignement.

Figure 3 : Diagramme de phase accompagné de la représentation schématique des chorégraphies collectives [1]

Finalement, le modèle des trois cercles concentriques propose une première explication du comportement des poissons à l'allure si organisée : il serait en fait dû au fait que les animaux adaptent constamment les rayons des différentes zones autour d'eux à leur environnement et à leurs congénères. Ce serait ainsi que naissent les chorégraphies aquatiques des poissons.

3.1.2. LE MODELE DES SIX VOISINS

Le modèle précédent a cependant montré ses limites de deux manières. Tout d'abord, si l'on modélise une attaque de prédateur sur un groupe de poissons, avec le modèle des trois cercles concentriques, le groupe se fragmente, ce qui n'est pas observé expérimentalement.

La deuxième mise en évidence des limites du modèle précédent a été mise en avant en 2007 par une équipe de l'université de Rome, qui a observé et filmé des vols d'étourneaux – qui ont un comportement analogue à celui des bancs de poissons – et a comparé cela à la modélisation du modèle des cercles concentriques. En comparant les résultats des deux simulations, ils ont réalisé que cela ne concordait pas.

L'équipe de recherches a alors analysé comment s'organisait le banc et où se plaçaient les oiseaux les uns par rapport aux autres. Ils ont relevé la position de chaque voisin pour chaque oiseau et ont pu dégager des motifs. La position moyenne dans l'espace du premier plus proche voisin pour chacun des étourneaux n'était pas homogène dans l'espace mais anisotropique, et de même pour les six premiers plus proches voisins. Cela permet de mettre en évidence le fait que les oiseaux (et poissons) ne se placent pas n'importe où dans l'espace, mais qu'ils sont influencés par rapport aux autres.

Cependant, ces observations ne suffisent pas à généraliser une valeur de limite. Nous pouvons penser que le nombre ou la densité d'oiseaux ou de poissons dans un banc à son

importance. En réitérant leurs observations sur des groupes de tailles différentes, les chercheurs ont découvert que cette valeur restait constante.[1]

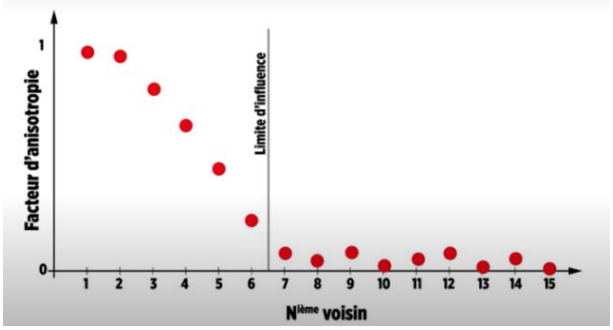


Figure 5 Graphique du phénomène représentant le facteur d'anisotropie en fonction du nième voisin [1]

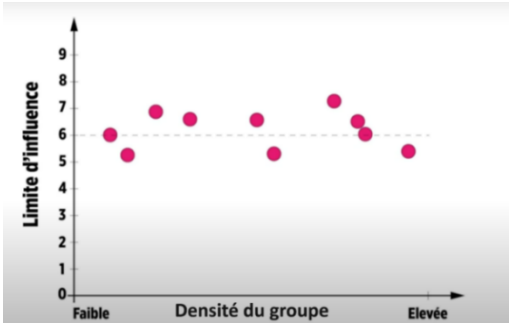


Figure 4 Graphique de la limite d'influence sur les poissons en fonction de la densité du groupe [1]

Cette étude a permis de montrer que chaque oiseau est influencé par ses six plus proches voisins, peu importe la distance avec ces derniers ou sa distance avec les autres, puisqu'à partir du septième voisin, il pouvait être n'importe où dans l'espace. Ils ont aussi remarqué que le nombre de voisins influents ne varie pas selon la taille du banc.

Afin de vérifier cela, ils ont comparé de nouveau avec les observations de bancs d'étourneaux : ils ont repris les équations d'attraction, alignement et répulsion précédentes mais ne les ont appliquées qu'aux six plus proches voisins. Ce modèle permet bien de garder une cohésion lors de l'attaque d'un prédateur et d'assurer des mouvements ordonnés d'un grand groupe d'oiseaux. Si un oiseau change de direction, cela s'impose à tout le reste du groupe par propagation.

Après avoir continué les recherches, il a été prouvé que c'est en réalité les six plus proches voisins visibles qui ont de l'influence. En effet, si un poisson est derrière celui étudié, ou caché par un autre poisson, il n'influence par celui-ci et ne sera donc pas pris en compte. [4]

3.1.3. NOTRE MODELE

Nous avons décidé, lors de l'élaboration de notre programme, de nous inspirer des deux modèles des cercles concentriques et des plus proches voisins afin de garder le meilleur de chacun d'entre eux. Nous avons donc utilisé le principe des six voisins en l'intégrant aux phases d'alignement et d'attraction. Lors de la phase de répulsion, notre modèle ne prend en compte que le plus proche voisin afin d'éviter les collisions. Ce modèle est plutôt réaliste : lorsqu'il y a beaucoup d'individus dans les cercles d'alignement et d'attraction, un individu ne peut pas suivre en simultané l'ensemble des poissons. Il n'en suit que six et uniquement les plus proches. À l'inverse, lorsque deux individus essayent de ne pas se toucher, ils cherchent seulement à s'éviter et arrêtent de prendre en compte le reste du banc.

Nous avons décidé de coder en langage Python car ce langage de programmation présente des bibliothèques très complètes qui permettent l'affichage de graphiques. De plus, coder en langage Python est légèrement plus rapide que de coder en Pascal par exemple.

Pour modéliser notre modèle, on décide de se placer dans un domaine du plan de dimensions variables. On positionne un certain nombre de poissons dans ce domaine auxquels on attribue une vitesse. Puis à chaque pas de temps, on étudie chaque poisson pour adapter sa trajectoire. Pour cela, on a besoin d'exploiter les trois notions suivantes : la répulsion, l'alignement et l'attraction, détaillées ci-dessous.

3.2. REPULSION

3.2.1. CONDITIONS D'APPLICATION DE LA REPULSION

Comme dit précédemment, la répulsion appliquée à un poisson d'étude se traduit par la déviation de ce poisson pour éviter la collision avec son plus proche voisin. Cela se manifeste en réalité par un changement de direction de son vecteur vitesse. Ce changement s'opère si plusieurs conditions sont réunies.

NB : On utilise ci-dessous la notion de cône de répulsion, dont voici l'explication. On place deux poissons P_i et P_j (son plus proche voisin) dans le plan. On représente autour de chacun d'eux leur espace vital par un cercle de rayon $R_{rép}$ dans lequel l'autre poisson ne doit pas rentrer. On définit un cône de répulsion représenté en hachuré sur la figure correspondant au triangle incomplet formé par les deux segments - ayant pour origine le poisson P_i - tangents au cercle délimitant l'espace vital de P_j.

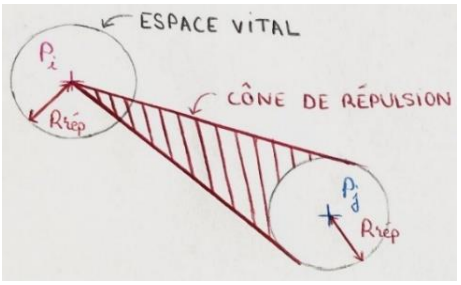


Figure 6 : Illustration du cône de répulsion

Condition 1 :

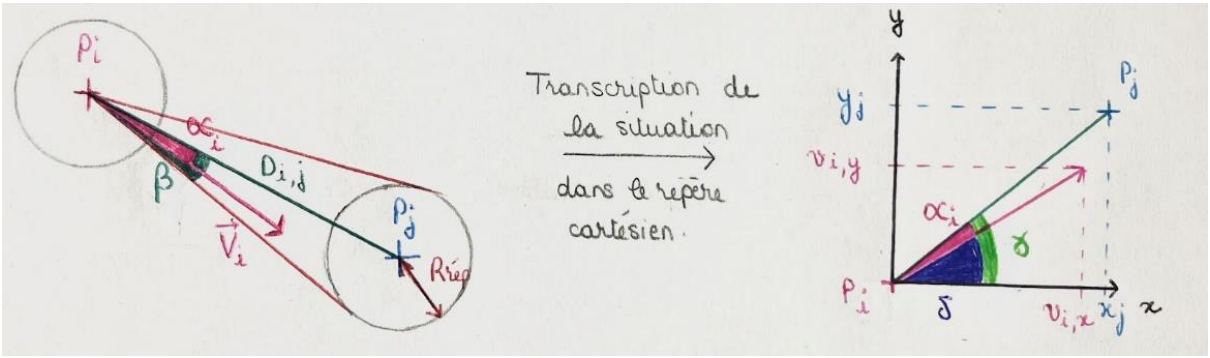


Figure 7 : Illustration de la condition 1 pour la répulsion

La répulsion est envisageable lorsque le poisson étudié P_i se dirige vers son plus proche voisin P_j. Cela se traduit par le fait que le vecteur vitesse $\vec{V_i}$ de P_i se trouve dans son cône de répulsion. En termes d'angles, le vecteur $\vec{V_i}$ se trouve dans le cône de répulsion si $\boldsymbol{\alpha_i < \beta}$, avec α_i l'angle entre le segment reliant P_i et P_j et le vecteur vitesse $\vec{V_i}$, et β l'angle entre le même segment reliant P_i et P_j et un des bords du cône de répulsion de P_i.

Or $\beta = arctan\left(\frac{Rrép}{Di,j}\right)$ avec $R_{rép}$ le rayon de répulsion du cercle entourant P_j et $D_{i,j}$ la distance entre P_i et P_j. Et $\alpha_i = \gamma - \delta = arctan\left(\frac{v_{i,y}}{v_{i,x}}\right) - arctan\left(\frac{y_j - y_i}{xj - xi}\right)$ avec $\vec{V_i} = \begin{pmatrix} v_{i,x} \\ v_{i,y} \end{pmatrix}$, $\overrightarrow{OP_i} = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$ et $\overrightarrow{OP_j} = \begin{pmatrix} x_j \\ y_j \end{pmatrix}$.

Condition 2 :

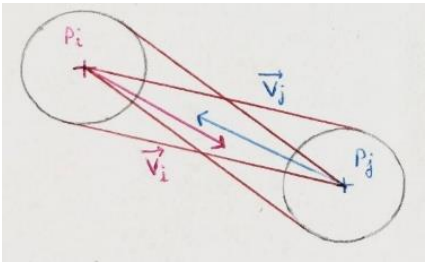
La répulsion est envisageable lorsque P_j, le plus proche voisin du poisson d'étude P_i, se dirige vers P_i. Cela se traduit par le fait que le vecteur vitesse $\vec{V_j}$ de P_j se trouve dans son cône de répulsion. En termes d'angles, le vecteur $\vec{V_j}$ se trouve dans le cône de répulsion si $\boldsymbol{\alpha_j < \beta}$, avec α_j l'angle entre le segment reliant P_i et P_j et le vecteur vitesse $\vec{V_j}$ et β l'angle entre le même segment

reliant P_i et P_j et un des bords du cône de répulsion de P_j. Donc de manière analogue à la condition 1 : $\beta = arctan(\frac{Rrép}{Di,j})$ et $\alpha_i = arctan\left(\frac{v_{j,y}}{v_{j,x}}\right) - arctan\left(\frac{y_i - y_j}{xi - xj}\right)$.

Condition 3 :

La répulsion est envisageable lorsque la collision entre P_i et P_j est imminente. Cela se traduit par le fait que le nombre de pas de temps avant la collision de P_i avec son plus proche voisin (noté n_{dc}) est faible, autrement dit que le vecteur vitesse $\vec{V_i}$ de P_i est relativement grand par rapport à la distance $D_{i,j}$ entre les deux poissons. En d'autres termes, la répulsion est envisageable si $\mathbf{n_{dc} < 5}$, avec $n_{dc} = \frac{D_{i,j}}{|| \vec{V_i} || \times \Delta t}$ avec $|| \vec{V_i} || = \sqrt{{v_{i,x}}^2 + {v_{i,y}}^2}$.

Déclenchement de la répulsion :

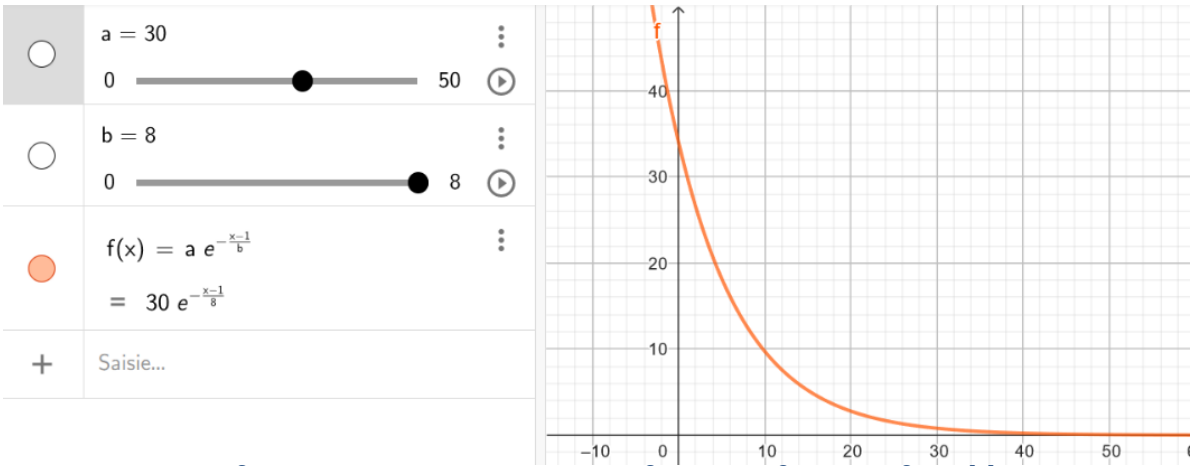


Dans notre programme, la répulsion s'applique lorsque les conditions 1, 2 et 3 sont toutes réunies. Cela se traduit par le fait que les deux poissons se dirigent l'un sur l'autre, et que la collision est imminente. (Conditions codées en *Annexe 1*)

Figure 8 : Illustration de la répulsion avec toutes les conditions

3.2.2. CHANGEMENT DE DIRECTION

Lorsque la répulsion est enclenchée, le poisson concerné doit changer de direction pour éviter la collision avec son plus proche voisin. Nous avons donc élaboré une fonction prenant en paramètre d'entrée le nombre de déplacement avant la collision (n_{dc}). Nous souhaitions que plus n_{dc} est petit, plus le changement d'angle est brutal. Nous avons également considéré que les poissons ne peuvent pas changer trop brusquement de direction, puisqu'en effet, lorsqu'un être vivant tente d'éviter une personne, il s'y prend assez à l'avance pour éviter d'avoir à réaliser un virage trop brusque pour éviter ce qu'il y a devant lui. Nous avons donc modélisé cela en définissant un angle de répulsion maximal de 30°, angle de déviation le plus grand que les poissons peuvent faire. Puis nous avons utilisé une fonction exponentielle décroissante qui donne l'angle de changement de direction que va faire le poisson en fonction de la distance à laquelle il se trouve de son obstacle. Afin de vérifier son fonctionnement et de définir le coefficient devant cette fonction, nous l'avons représenté sur Geogebra et nous avons fait varier les paramètres afin de trouver ce qui correspond le mieux à la réalité. Ainsi, plus un poisson est loin, moins il va dévier brusquement.



3.2.3. RENDU FINAL DU CODAGE DU PHENOMENE DE LA REPULSION

Le rendu final du codage du phénomène de répulsion est disponible au lien suivant : https://drive.google.com/file/d/18TG3MQa2V-9WIDAJUIIDKvsQUw3T0EKF/view?usp=drive_link.

On remarque que les poissons s'évitent au maximum, et occupent tout l'espace qui leur est réservé afin de s'éloigner les uns des autres. Leur comportement est cohérent car pour l'instant, il n'y a pas d'attraction, ni d'alignement. Lorsque les poissons viennent à se croiser, on observe une modification de leur trajectoire pour tenter de s'éviter. Néanmoins, notre méthode de répulsion a ses limites, et on peut voir leurs conséquences à l'écran : certains poissons se frôlent malgré toutes les conditions imposées. Cela peut s'expliquer par deux choses : d'abord, la répulsion s'effectue uniquement avec l'individu le plus proche, ce qui peut entraîner des collisions lorsque qu'un grand nombre d'individus se retrouvent au même endroit, puisque la manœuvre d'évitement de chaque poisson peut l'envoyer sur un autre poisson qui n'était pas son plus proche voisin au pas de temps précédent. Cependant, ce défaut peut tout à fait se retrouver dans la nature : il peut arriver que des poissons se foncent dedans par manque d'anticipation. Le second problème de notre modélisation concerne les bords du bassin. En effet, le rebond sur les côtés du bassin prédomine sur la répulsion car le poisson est immédiatement renvoyé dans le sens inverse sans que cela ne soit prévu. Si un autre individu se situe à cet endroit, l'évitement ne peut pas être anticipé. Ce genre de situation arrive tout de même moins souvent lorsque les paramètres d'alignement et d'attraction sont pris en compte, car les déplacements sont influencés pour éviter que les individus se dirigent les uns sur les autres mais plutôt pour qu'ils s'alignent à une certaine distance.

Afin de vérifier la cohérence de notre programme, nous avons effectué plusieurs tests qui consistaient à faire varier les paramètres utilisés par le programme, et d'observer comment ils impactaient le mouvement des poissons. (Code de la répulsion disponible en *Annexe 2*)

Nous avons alors pu observer que lorsque l'on baisse l'angle maximal de répulsion à quelques degrés, la répulsion devient presque invisible : en effet, les poissons n'ont pas le temps de dévier leur trajectoire pour éviter la collision que déjà ils ont dépassé le poisson gênant. C'est comme si la répulsion n'existait plus. En revanche, lorsqu'on augmente le rayon maximal de répulsion, cette dernière devient plus abrupte, et sensiblement plus efficace : le nombre de frôlements diminue. On observe le même phénomène lorsqu'on fait varier le nombre de déplacements avant la collision n_{dc}. En effet, on observe beaucoup plus de frôlements, voire de collisions lorsqu'on baisse n_{dc} que lorsqu'on l'augmente. C'est cohérent avec notre codage : baisser n_{dc} a pour conséquence de retarder le déclenchement de la répulsion, et donc d'augmenter le risque de contact.

La variation des autres paramètres utilisés dans la répulsion (Rrép et b (le tau de répulsion utilisé dans la fonction du changement d'angle exposée dans le 3.2.2.) n'offre pas de différence discernable à l'œil nu.

3.3. ALIGNEMENT

L'alignement d'un poisson par rapport aux autres se traduit par l'adoption d'une direction similaire à celle de ses voisins les plus proches. Les poissons vivent en communauté et évoluent en banc afin de se protéger des prédateurs. C'est pour cette raison qu'il est important que les individus restent à une distance raisonnable des uns des autres. L'alignement permet alors que le poisson ne s'éloigne pas ou ne se rapproche pas trop de ses voisins.

3.3.1. CONDITION D'APPLICATION

Au sein de notre programme, la fonction d'alignement est appliquée par défaut, c'est-à-dire que si le poisson n'est pas en phase de répulsion et qu'il se situe à une distance inférieure au rayon d'attraction, alors l'alignement est appliqué. La littérature ne donne pas de distance précise pour ce rayon et la mise à l'échelle sur ordinateur est difficile à évaluer. Pour déterminer le rayon d'attraction, nous avons tout d'abord décidé arbitrairement d'une valeur (120 pixels) puis nous avons testé différentes valeurs pour trouver celle qui fonctionne le mieux. En testant avec un rayon de 15, nous avons pu remarquer que les poissons formaient des petits groupes et avaient moins de mouvement de groupe qu'avec un rayon d'attraction de 120. Nous avons alors décidé de garder 120 en rayon d'attraction pour pouvoir conserver ce mouvement de groupe. Néanmoins il n'y a pas de différence notable lorsque le rayon est plus large.

3.3.2. PRINCIPE DE FONCTIONNEMENT

Le principe de notre méthode est de déterminer la vitesse globale des six voisins les plus proches et de l'attribuer au poisson étudié. Pour la suite, la vitesse moyenne du groupe sera considérée comme la vitesse cible.

Dans un premier temps, nous avons calculé la vitesse moyenne des six voisins les plus proches, il s'agit en fait d'une vitesse moyenne pondérée par la distance respective de chaque individu avec le poisson étudié. C'est à dire qu'un poisson plus proche du poisson étudié a plus d'influence sur l'orientation de ce dernier, qu'un poisson se trouvant plus éloigné.

La formule résultante est donc : $\overrightarrow{V_{cible}} = \frac{\sum_{j=1}^{6} \overrightarrow{V_j} \times \frac{1}{l_{ij}}}{\sum_{j=1}^{6} \frac{1}{l_{ij}}}$, avec l_{ij} la distance entre le poisson considéré et son voisin j.

Ce vecteur vitesse moyenne est alors considéré comme le vecteur cible, cela signifie que dans l'idéal nous souhaiterions que le poisson étudié adopte cette nouvelle vitesse au pas de temps suivant. Néanmoins ce n'est pas aussi simple que cela, il faut tenir compte de la vitesse actuelle du poisson car en pratique celui-ci ne peut pas changer brutalement d'orientation ou de vitesse de déplacement. Notre objectif a alors été de partir de ce vecteur cible et de l'adapter de façon à ce qu'il soit cohérent d'un point de vue pratique. Pour cela il a fallu soumettre le vecteur cible à deux conditions, une sur la vitesse et une sur le changement d'orientation.

Concernant l'adaptation du changement d'orientation, nous avons défini une méthode théorique permettant d'effectuer ce changement [*Annexe 3*]. En pratique le résultat visuel sur le banc de poissons ne correspondait pas à ce que nous souhaitions et il est difficile pour nous de savoir s'il s'agit d'une erreur de codage ou de raisonnement étant donné qu'après avoir cherché à de multiples reprises nous ne sommes pas parvenus à repérer l'origine de notre problème. Finalement, notre programme contient uniquement l'adaptation sur la vitesse néanmoins les résultats restent cohérents.

3.3.3. AJUSTEMENT DE LA NORME DE LA VITESSE

Les poissons ou les oiseaux se déplacent à une vitesse constante en dehors des attaques de prédateurs. Pour répondre à cette problématique nous avons alors appliqué une norme de vitesse constante pour l'ensemble des poissons. Le vecteur cible a été normalisé grâce à l'application de la formule suivante. Notons $\overrightarrow{V_c}$ le vecteur vitesse cible et $\overrightarrow{V_{al}}$ le vecteur vitesse après l'adaptation de vitesse :

$$\overrightarrow{V_{al}} = \frac{\overrightarrow{V_c}}{\|\overrightarrow{V_c}\|} \times \|\overrightarrow{V_{cste}}\|$$

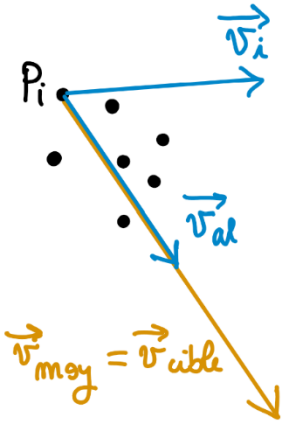


Figure 10: Illustration du principe d'alignement

3.3.4. RENDU FINAL DU CODAGE DU PHENOMENE D'ALIGNEMENT

En théorie, tous les poissons doivent avoir la même direction pour rester ni trop proches ni trop loin les uns des autres. On observe que les poissons suivent des trajectoires linéaires et que leur mouvement est uniforme. De plus, on remarque que certains poissons sont bloqués sur les bords avant de changer de direction. En effet, il est nécessaire qu'un nombre suffisant de poissons aient atteint les bords pour pouvoir changer définitivement leur orientation car c'est le voisin le plus proche qui a le plus d'influence dans notre code. Bien que les poissons forment des groupes, ils ne sont pas assez proches les uns des autres d'où l'importance de l'attraction. Enfin, puisque certains poissons se croisent encore, il est indispensable de coder une répulsion.

Le rendu final du codage du phénomène de l'alignement est disponible en suivant le lien suivant https://drive.google.com/file/d/1uCspXq05ktEX1OAx5fcG0hQGXohshmJ0/view?usp=drive_link, et le code est disponible en *Annexe 4*.

3.4. ATTRACTION

3.4.1. FONCTIONNEMENT GENERAL DE L'ATTRACTION

Nous avons modélisé les phénomènes de répulsion et d'alignement. Les poissons ne rentrent donc pas en collision et prennent la même direction. Néanmoins, si un poisson est éloigné des autres, il faudrait le rapprocher du groupe pour qu'il n'y ait qu'un seul banc. C'est pour cela que nous avons rajouté un phénomène d'attraction aux phénomènes déjà modélisés.

Nous avons décidé de définir l'attraction d'un poisson par rapport à une distance avec le centre de gravité de ses voisins. Le poisson va donc se diriger vers le centre de masse des 6 voisins, ce qui lui permettra de rejoindre le groupe.

L'idée derrière cela est de calculer le centre de gravité des 6 voisins du poisson grâce aux formules suivantes : $x_g = \frac{\sum_{j=1}^{6} x_j}{6}$ et $y_g = \frac{\sum_{j=1}^{6} y_j}{6}$ avec x_g et y_g les positions du barycentre, j l'indice des voisins.

Ensuite, on utilise un vecteur vitesse cible comme dans l'alignement. Le vecteur vitesse cible est le vecteur entre le poisson et le barycentre. Ce vecteur est le vecteur vitesse "idéal", le vecteur que le poisson prendrait dans le meilleur des mondes. On a donc $\overrightarrow{v_c} = (\frac{x_g - x_p}{\alpha}, \frac{y_g - y_p}{\alpha})$, avec (x_p, y_p) les coordonnées du poisson et α le pas de temps qui correspond à une boucle du programme. Chaque boucle dure 1 seconde dans le programme.

Encore une fois, puisque nous avons considéré que les poissons ne peuvent changer brusquement de direction dépendant de l'angle auquel ils doivent se tourner, nous avons adapté la vitesse cible pour que la vitesse finale se trouve entre la vitesse cible et la vitesse actuelle du poisson. On utilise les mêmes formules que dans l'alignement pour l'adaptation de la vitesse cependant, comme dit précédemment, cette partie n'a pas abouti à de bons résultats et ne fait donc pas partie du code. *[Annexe 3]*
Enfin, nous avons divisé la vitesse finale par la norme du vecteur vitesse cible et multiplié par la vitesse constante pour normaliser le résultat :

$$\overrightarrow{v_{att}} = \frac{\overrightarrow{v_c} \times \|\overrightarrow{v_{cste}}\|}{\|\overrightarrow{v_c}\|}$$

Ainsi, le poisson mesure la distance avec le centre de gravité de ses voisins et prend la direction vers celui-ci s'il est trop loin. (Code en *Annexe 5*)

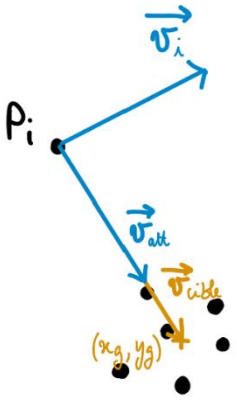


Figure 11: Illustration du principe d'attraction

3.4.2. CONDITIONS POUR APPLIQUER L'ATTRACTION

Le test pour savoir si on applique l'attraction se fait donc en fonction du barycentre calculé auparavant. À partir de ce calcul, on peut déterminer la distance entre les 2 et donc savoir si le poisson doit ou non se rapprocher. On calcule la distance entre le barycentre et le poisson avec : $d = \sqrt{(x_g - x_p)^2 + (y_g - y_p)^2}$ avec p l'indice du poisson. Si celui-ci est plus grand ou égal au rayon d'attraction, on applique l'attraction. (Code en *Annexe 6*)

3.4.3. RENDU FINAL DU CODAGE DU PHENOMENE DE L'ATTRACTION

Tous les poissons doivent théoriquement se rapprocher les uns des autres afin de former des groupes. On observe que les poissons ne forment qu'un. Lorsque certains s'éloignent, ils retournent en direction de leur groupe de six voisins les plus proches. Le changement de direction des poissons s'opère par un demi-tour qui serait préférable d'adapter pour être plus proche de la réalité. C'est pourquoi cette adaptation fait partie de nos perspectives d'amélioration du programme. L'efficacité de l'attraction varie selon la constante "rayon_att". En effet, plus ce rayon est petit, plus les poissons restent proches les uns des autres.

En revanche, le mouvement des poissons reste diffus et limité, c'est pourquoi l'alignement est important pour guider les poissons dans une direction. On observe également des collisions entre les poissons ce qui montre que la répulsion n'est pas négligeable.

Le rendu final du codage du phénomène de l'attraction est disponible en suivant le lien suivant : https://drive.google.com/file/d/1g_HcEwlKXteq-seCiYebd3QLVAkOFMZb/view?usp=drive_link.

3.5. MISE EN COMMUN DE TOUS LES PHENOMENES

3.5.1. EXPLICATION DU FONCTIONNEMENT GENERAL DU PROGRAMME

Pour le bon fonctionnement du programme, nous avions besoin de certaines bibliothèques Python déjà existantes, que nous avons donc commencé par importer. Puis nous avons posé un certain nombre de constantes avant de commencer le cœur du programme afin de pouvoir les modifier facilement par la suite pour effectuer nos tests. *(Annexe 7)*.

Afin de résumer comment le programme principal s'organise, le lien entre chacune des fonctions implémentées et leurs entrées et sorties, nous avons réalisé une analyse descendante, que vous pouvez retrouver en *Annexe 8.*

Le programme principal (*Annexe 9*) consiste à d'abord initialiser les poissons et le bassin, puis à générer l'affichage, avant de répéter le nombre de fois souhaitées une boucle qui calcule la distance entre les différents poissons et leurs plus proches voisins. Puis il calcule leur mouvement à l'aide des fonctions de répulsion, d'alignement et d'attraction. Il initialise ensuite la position de chaque poisson et l'affiche. Voici plus en détails comment chacune des parties fonctionne :

- ini_banc (*Annexe 10*) : Cette fonction permet de créer un tableau contenant le nombre de poissons voulus, et pour chacun de lui associer une vitesse (norme et direction) et une position comprise dans le bassin défini par les longueurs fixées dans les constantes.
- affichagepoint (*Annexe 11*) : Pour chacun des poissons contenus dans le banc, cette partie du programme va l'afficher en prenant en compte les délimitations fixées pour la taille du bassin.
- distance_entre_poisson (*Annexe 12*) : Cela calcule la distance du poisson considéré avec tous les autres, met ces valeurs dans un tableau puis classe le tableau de la distance la plus courte à la plus grande pour trouver ces plus proches voisins, qui vont l'influencer par la suite.

- Mouvement (*Annexe 13*): Cette partie permet d'organiser quelles fonctions doivent être appliquées entre la répulsion, l'alignement et l'attraction en fixant un ordre de priorité et en vérifiant les conditions d'utilisation, puis calcule la vitesse qu'aura le poisson selon cela. Tout d'abord, on vérifie si la répulsion est nécessaire, puisque le réflexe premier d'un être vivant est d'éviter la collision avec ses voisins. Si besoin, elle est appliquée et la position du poisson est adaptée uniquement par rapport à cela, sinon, on regarde si l'attraction est nécessaire, et on l'applique si oui, car on cherche à se rapprocher des autres si l'on est trop loin. Pour finir, si ni la répulsion ni l'attraction n'a ses conditions vérifiées, on aligne le poisson sur ses voisins, car ils sont à une distance considérée comme bien pour se déplacer.
- positionpoisson (*Annexe 14*) : calcule la prochaine position du poisson pour pouvoir l'afficher, en fonction de la direction de sa vitesse au temps précédent.

Lorsqu'on lance le programme, on passe donc par toutes ces étapes. Alors que les poissons sont désordonnés au début, on observe peu à peu un mouvement d'ensemble se dégager au bout de quelques répétitions. Vous pouvez notamment le voir sur la vidéo suivante : https://drive.google.com/file/d/1oXhFvI8HqYSqpFR_461cjN1PETj6CfA_/view?usp=drive_link.

Au tout début, les poissons partent tous dans des directions différentes et sont placés sur toute la surface du bassin. Puis ils se regroupent par petits groupes qui s'organisent et se déplacent ensemble. La répulsion n'est pas très visible puisque la taille du bassin est très grande devant le rayon des poissons. Toutefois, on observe bien l'attraction puisque les poissons se rapprochent les uns des autres, ainsi que l'alignement puisque chaque groupe de poissons va dans la même direction. Puis au bout d'un temps assez conséquent, on voit que tous les poissons se sont organisés entre eux, qu'ils vont tous dans la même direction, en gardant la même distance entre eux. On a bien la visualisation d'un banc.

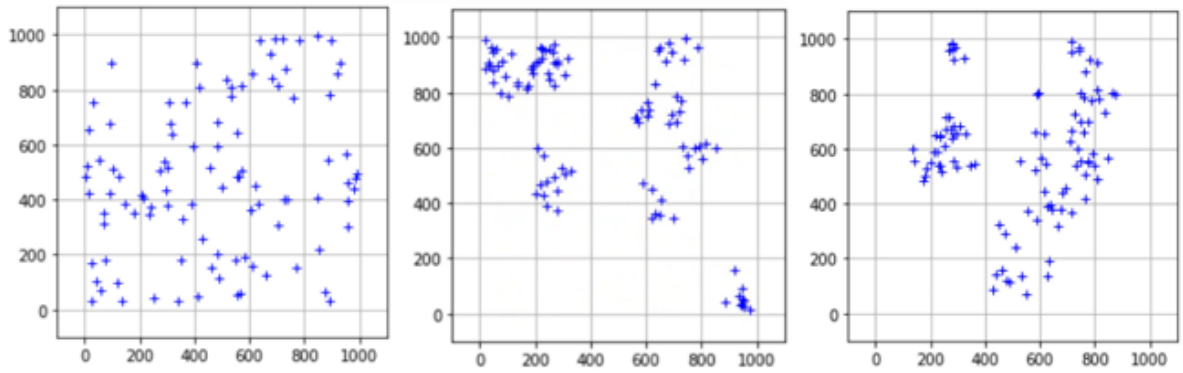


Figure 12: Evolution du mouvement des poissons lorsque l'on applique toutes les conditions

3.5.2. VARIATION DU NOMBRE DE VOISINS

Comme évoqué dans la partie 3.1.2., les scientifiques ont déterminé par l'expérience le nombre idéal de voisins à prendre en compte afin de se rapprocher le plus de la réalité observée. En faisant varier le nombre de voisins considérés dans notre programme, nous avons pu confirmer ce nombre de six voisins.

On observe en effet qu'en abaissant le nombre de voisins à deux, les poissons ont du mal à constituer un banc, et restent par petits groupes dispersés. Au sein de chaque groupe, les poissons restent collés et ont du mal à effectuer des mouvements d'ensemble. Etant donné que seule la position des deux voisins les plus proches est prise en compte, ce résultat est cohérent puisque les poissons n'ont pas assez de repères pour tous suivre le même mouvement. Ils se retrouvent donc à suivre uniquement les poissons très proches d'eux et forment ainsi plein de petits groupes. Le résultat de la simulation est disponible au lien suivant : https://drive.google.com/file/d/1uLBoY2P6FTKnUJWwAid-Ki6XfYh1YJic/view?usp=drive_link.

Inversement, en augmentant le nombre de voisins à dix, on remarque que les poissons ont tendance à tous se rapprocher les uns des autres dès le début, puis restent en groupe d'une dizaine d'individus, sans parvenir à mettre en place un mouvement commun. Ils ont du mal à se déplacer dans l'ensemble de l'espace et restent principalement vers le centre. Cela s'explique par le fait que chaque déplacement est influencé par celui des dix plus proches voisins, et que par conséquent le mouvement d'un ou deux poissons dans une nouvelle direction n'a pas assez d'impact pour faire changer l'ensemble du groupe. Le résultat de la simulation est disponible au lien suivant : https://drive.google.com/file/d/1IWW1ziFlusHwE38FGFp1i4eqa2R8ND8R/view?usp=drive_link.

4. CONCLUSIONS ET PERSPECTIVES

Ainsi, la réalisation de ce projet a nécessité de faire de la recherche – afin d'étudier les modèles existants et les mouvements réels des animaux –, de la réflexion – puisqu'il a fallu imaginer le raisonnement des poissons qui s'attirent ou se repoussent et créer des fonctions mathématiques retranscrivant les mouvements imaginés. Il aura ensuite fallu définir les conditions d'applications de chacune des fonctions, leur ordre de priorité et il a été nécessaire de les coder et de les tester. Puis, nous avons analysé les modélisations créées et fait varier les paramètres afin d'étudier leur influence.

Ce projet a été enrichissant pour chacun d'entre nous, puisqu'il nous a permis de développer notre esprit critique, en essayant de voir si les programmes modélisaient bien les mouvements voulus ; notre réflexion pour modifier les fonctions mathématiques au mieux pour qu'elles correspondent à nos besoins ; et de nous confronter à la difficulté de créer un raisonnement scientifique en partant d'aucune base. De plus, la plupart d'entre nous souhaitent travailler dans des domaines très différents de l'informatique, ce projet nous a permis de développer des compétences informatiques et de découvrir un nouveau langage de programmation, ce qui sera utile dans notre futur, peu importe notre domaine d'étude. Pour finir, nous avons pu expérimenter le travail d'ingénieur – que l'on sera amené à faire dans le futur – en travaillant en équipe, en se répartissant les taches, en partageant les informations chaque semaine en se réunissant en classe et en coopérant.

Plusieurs points nous sont toutefois venus à l'esprit qu'il serait utile d'essayer d'améliorer pour développer encore plus ce projet et la modélisation déjà créée :

- On pourrait améliorer le code et surtout sa complexité en temps (temps nécessaire pour l'algorithme pour effectuer les calculs) afin de le rendre plus efficace et ainsi pouvoir le tester avec un plus grand nombre de poissons – ce qui est impossible à l'heure actuelle car le programme nécessite beaucoup de boucles de calcul et prend donc un temps assez conséquent.
- Il faudrait instaurer une répulsion sur les bords pour rendre le mouvement des poissons plus réaliste, puisqu'ils tournent et longent le bord et ne rebondissent pas dessus comme nous l'avons fait. Pour cela, nous pourrions réutiliser la fonction qui calcule la répulsion entre poissons et qui adapte leurs angles, mais en prenant en compte la distance entre le poisson et le bord du bassin.
- Par soucis de simplification du programme, nous avons considéré que tous les poissons avaient la même vitesse initiale et invariante. Il serait intéressant de lever cette simplification et de distribuer les vitesses des poissons en utilisant une répartition gaussienne centrée sur 12 m/s, la vitesse moyenne des sardines.
- Pour l'alignement et l'attraction, on pourrait mettre une adaptation d'angle au moment du changement de direction pour ne pas avoir de changements brusques, comme expliqué précédemment.

- De plus, il serait intéressant de le modéliser en 3D, puisque les poissons et étourneaux se déplacent dans l'espace et non dans le plan.
- On pourrait également coder l'apparition d'un prédateur, comme un requin par exemple, qui serait introduit dans le banc de poissons, ce qui amènerait les poissons à fuir. On pourrait ainsi analyser comment le changement de direction soudain de quelques poissons influence l'ensemble du banc.
- Pour finir, on pourrait assimiler le banc de poissons à un écoulement et faire une modélisation en eulérien. Dans tout notre projet, nous avons modélisé en lagrangien, en fonction de chaque particule. Il serait donc intéressant de faire une modélisation en décrivant le champ de vitesses qui associe à chaque point un vecteur vitesse.

Le code informatique complet est disponible en cliquant sur le lien suivant :

https://drive.google.com/file/d/1KKCLKS-LKxSe8i2v45qqJfjuuvGXJyHk/view?usp=drive_link

5. BIBLIOGRAPHIE

[1] lien internet : https://www.youtube.com/watch?v=Ch7VxxTBe1c (valide à la date du 29/05/2024).

[2] lien internet : https://www.youtube.com/watch?v=Eh7l4Gvx054 (valide à la date du 29/05/2024).

[3] lien internet : http://www.mehdimoussaid.com/quest-ce-que-la-fouloscopie/ (valide à la date du 29/05/2024).

[4] lien internet : https://theses.hal.science/tel-03329573v1/document (valide à la date du 29/05/2024).

[5] lien internet : https://www.geogebra.org/classic?lang=fr (valide à la date du 30/05/2024).

6. ANNEXES

ANNEXE 1 : Programmation des conditions d'utilisation de la répulsion

```
def test_cone( poisson1posx, poisson1posy, poisson2posx,poisson2posy, poisson1vx, poisson1vy, distanceauvoisin):
    """compare l angle max conique du poisson avec l'angle de la position repulser.
    Renvoie true si le poisson est dans le cone"""
    beta = atan((rayonpoisson/distanceauvoisin))
    alpha = atan(poisson1vy/poisson1vx) - atan((poisson2posy - poisson1posy)/ (poisson2posx - poisson2posy))
    return abs(alpha) < abs(beta)

def test_distance_avant_collision(distance, normevitesse):
    """renvoie true si les poissons vont se percuter dans un nombre de pas de temps"""
    return (distance / normevitesse)*pas_temps < 5

def test_repulsion(banc : list , num_p : int, tabvois : list)-> bool:
    """verifie toutes les conditions pour appliquer la repulsion sur le voisin le plus proche"""
    distanceauvoisin = dist_voisin(num_p, 0, tabvois)
    if test_cone(positionx(banc,num_p),positiony(banc,num_p), positionx(banc,num_voisin(num_p, 0, tabvois))\
    ,positiony(banc,num_voisin(num_p,0,tabvois)), vitessex(banc, num_p), vitessey(banc, num_p), distanceauvoisin):
        if test_distance_avant_collision(distanceauvoisin, norme_vitesse(vitessex(banc, num_p), vitessey(banc, num_p))):
            return True
    return False
```

ANNEXE 2 : Programme de répulsion

```
def anglefinalerepulsion (banc : list, num_p : int , tabvois : list)-> int:
    """calcul l'angle de repulsion qui doit être appliqué au poisson afin d'eviter un autre poisson.
    cet angle est atténué et recalculé afin de ne pas faire de changement de direction brusque"""
    ndc = dist_voisin(num_p, 0, tabvois)/norme_vitesse(vitessex(banc,num_p), vitessey(banc, num_p))
    if ndc<1:
        ndc = 1
    anglefutur = angle_repulsion_max*exp(-((ndc - 1)/tau_repulsion))
    return anglefutur
```

ANNEXE 3 : Adaptation du changement d'orientation

Il faut adapter l'angle de déviation afin que le poisson ne se retrouve pas dans le sens opposé en l'espace d'un pas de temps, en effet dans la vie réelle les poissons comme les humains changent d'orientation petit à petit, la rotation se fait progressivement. Il n'y a pas d'angle défini caractérisant la rotation maximale d'un individu puisque cela dépend aussi de sa vitesse. Nous avons donc décidé que le poisson ne pourrait pas effectuer de changement d'angle supérieur à 30° en un seul pas de temps. Cela nous paraissait cohérent au vu de nos mouvements en tant qu'humain. L'allure de la fonction erf convenait à ce que nous recherchions en termes d'ajustement.

Nous avons effectué des tests sur Géogébra en faisant varier les différents paramètres de la fonction afin d'obtenir la forme que nous souhaitions. Nous sommes parvenus à l'équation suivante : $f(x) = 15\,\mathrm{erf}\left(\frac{x}{25} - 2\right) + 15$.

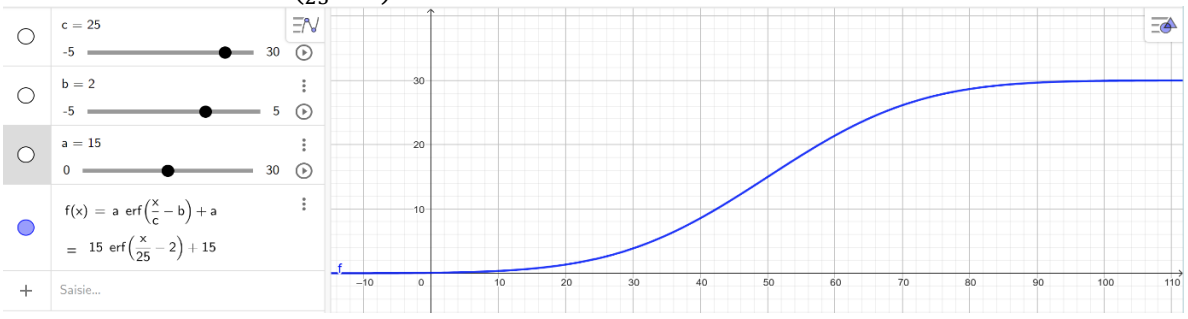


Figure 13 : Modélisation de la fonction d'erreur utilisée

Ainsi lorsque l'angle de déviation cible est supérieur ou égal à 90°, le poisson change directement de 30°. Si l'angle de déviation cible est plus faible alors la déviation se fait plus progressivement. Par exemple si l'angle cible est de 30° le poisson va changer de seulement 4° en un pas de temps, puis le calcul va se répéter pour l'étape suivant jusqu'à ce que l'angle de déviation cible soit nul.

ANNEXE 4 : Programme de l'alignement

```
def alignement(banc : list, tabvois : list, poisson : int) -> list :
    """fonction qui renvoie les vitesses après alignement"""
    vitesse_alignement = []
    """tableau avec les vitesses après le pas"""
    vitesse_alignement = [[None] * 2 for i in range(nombrepoisson)]

    somme_vitesse_x = 0
    somme_vitesse_y = 0
    somme_distance = 0
    for voisin in range(nombre_de_voisin):
        v_voisin_x = vitessex(banc, num_voisin(poisson, voisin, tabvois))
        v_voisin_y = vitessey(banc, num_voisin(poisson, voisin, tabvois))
        distance_voisin = dist_voisin(poisson, voisin, tabvois)
        somme_vitesse_x += v_voisin_x * (1/distance_voisin)
        somme_vitesse_y += v_voisin_y * (1/distance_voisin)
        somme_distance += 1/distance_voisin
        """calcule la vitesse moyenne du poisson par rapport à ses 6 voisins"""

        vitesse_moyenne_x = somme_vitesse_x/somme_distance
        vitesse_moyenne_y = somme_vitesse_y/somme_distance
        v_moy = norme_vitesse(vitesse_moyenne_x, vitesse_moyenne_y)
        v_moy_a_x=(vitesse_moyenne_x*vitesse)/v_moy
        v_moy_a_y=(vitesse_moyenne_y*vitesse)/v_moy
        vitesse_alignement[poisson][0] =v_moy_a_x
        vitesse_alignement[poisson][1]=v_moy_a_y
    return vitesse_alignement
```

ANNEXE 5 : Programme de l'attraction

```
def attraction(banc : list, tabvois : list, poisson : int) -> list :
    """permet d'attirer les poissons si celui-ci valide le test_attraction.
    l'attraction se fait par rapport a ses voisins. Le programme calcule la direction moyenne vers
    laquelle le poisson doit aller"""
    vitesse_att = []
    vitesse_att = [[None]*2 for i in range(nombrepoisson)]
    sum_coord_x = 0
    sum_coord_y = 0
    for voisin in range(nombre_de_voisin) :
        sum_coord_x += positionx(banc, num_voisin(poisson, voisin, tabvois))
        sum_coord_y += positiony(banc, num_voisin(poisson, voisin, tabvois))
    x_g = sum_coord_x/nombre_de_voisin
    y_g = sum_coord_y/nombre_de_voisin
    vc_x = (x_g - positionx(banc, poisson))/pas_temps
    vc_y = (y_g - positiony(banc, poisson))/pas_temps
    vx = vitessex(banc, poisson)
    vy = vitessey(banc, poisson)
    norme_vc = norme_vitesse(vc_x, vc_y)
    va_x=(vc_x*vitesse)/norme_vc
    va_y=(vc_y*vitesse)/norme_vc
    vitesse_att[poisson][0]=va_x
    vitesse_att[poisson][1]=va_y

    return vitesse_att
```

ANNEXE 6 : Programme de vérification des conditions de l'attraction

```python
def test_attraction(banc : list, num_p : int, tabvois : list) -> bool :
    """permet de tester si le poisson est elligible a l'attraction"""
    sum_coord_x = 0
    sum_coord_y = 0
    for voisin in range(nombre_de_voisin) :
        sum_coord_x += positionx(banc, num_voisin(num_p, voisin, tabvois))
        sum_coord_y += positiony(banc, num_voisin(num_p, voisin, tabvois))
    x_g = sum_coord_x/nombre_de_voisin
    y_g = sum_coord_y/nombre_de_voisin
    if distance(x_g, y_g, positionx(banc, num_p), positiony(banc, num_p)) >= rayon_att :
        return True
    else :
        return False
```

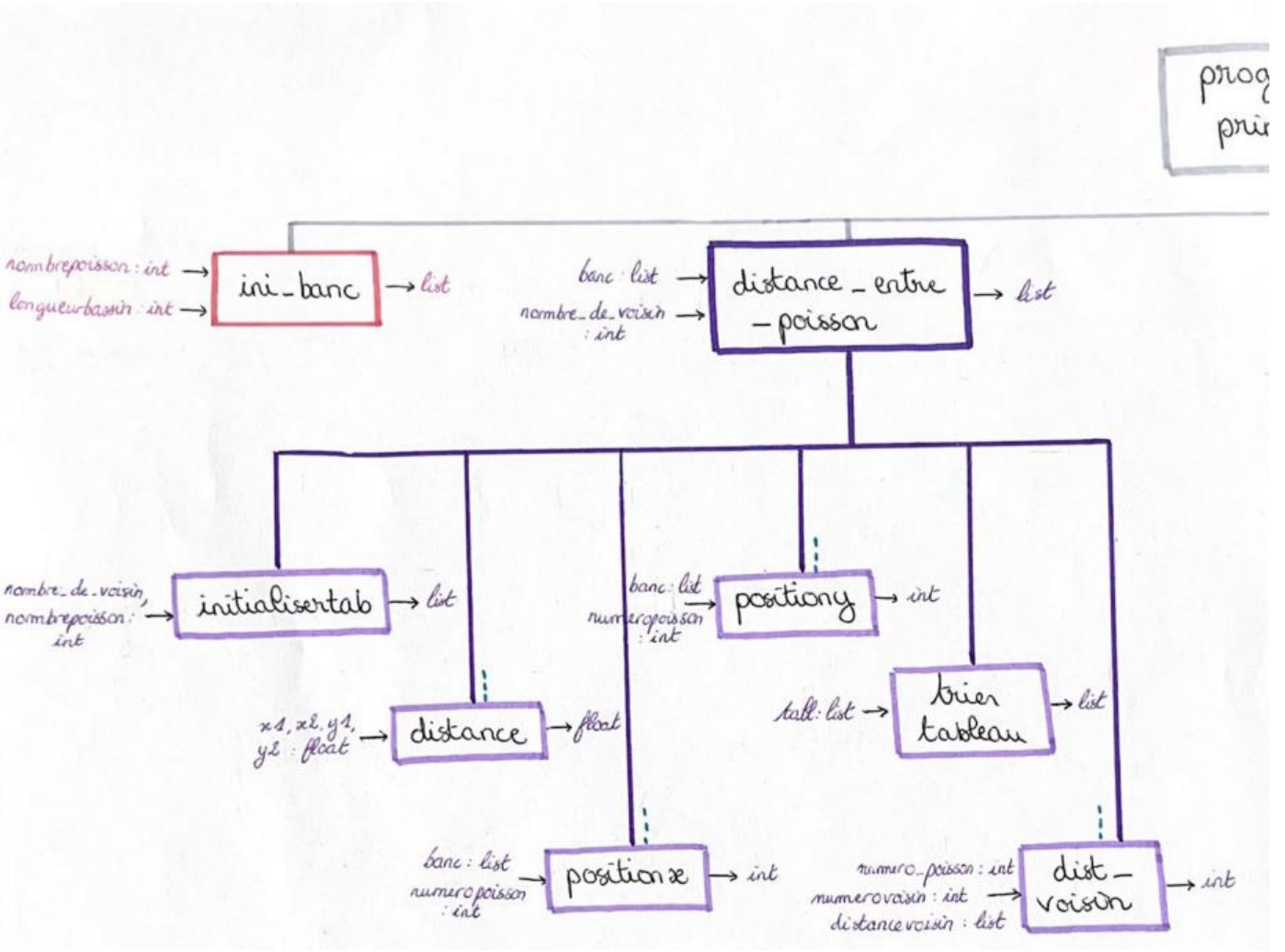
ANNEXE 7 : Constantes et bibliothèques utilisées

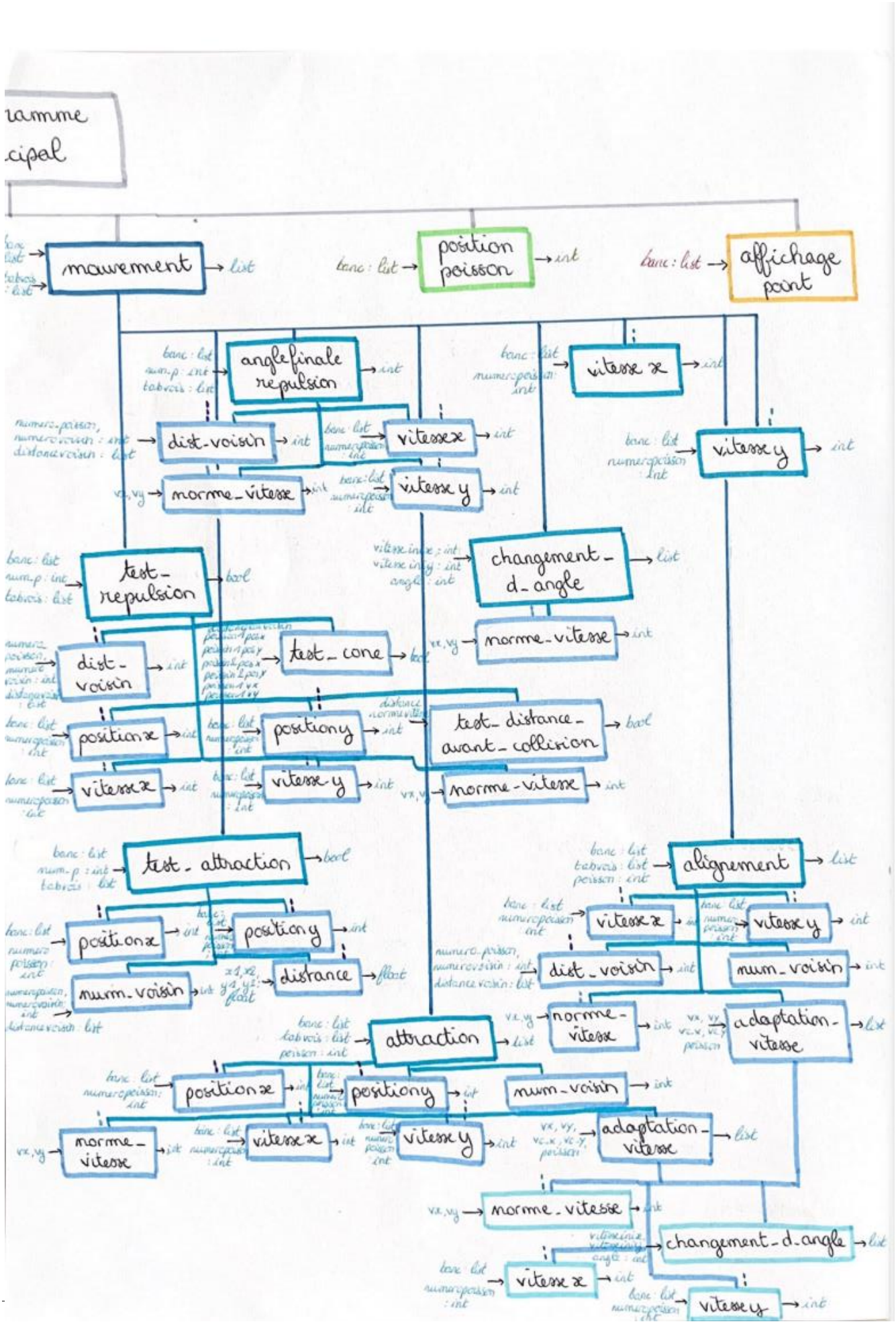
```python
# -*- coding: utf-8 -*-
import string
from tkinter import *
from random import choice, gauss, random
from math import*
import matplotlib.pyplot as plt

from matplotlib.figure import Figure

#variables
nombrepoisson = 100
nombre_de_voisin = 6
longueurbassin = 1000
rayon_att = 120
vitesse = 12
pause = 0.1
limiteaxe = (-100, longueurbassin+100)
rayonpoisson = 15
angle_repulsion_max = 30 * pi/180
angle = 30
tau_repulsion = 3
pas_temps = 1
nombre_de_tours = 100
```

ANNEXE 8 : Analyse descendante du programme




ramme
cipal
mouvement
position poisson
affichage point
angle finale repulsion
vitesse x
dist-voisin
vitesse x
vitesse y
norme_vitesse
vitesse y
test_ repulsion
changement_ d_angle
dist_ voisin
test_ cone
norme_vitesse
position x
position y
test_ distance_ avant_ collision
vitesse x
vitesse y
norme_vitesse
test_ attraction
alignement
position x
position y
vitesse x
vitesse y
num_voisin
distance
dist_voisin
num_voisin
attraction
norme_ vitesse
adaptation_ vitesse
position x
position y
num_voisin
norme_ vitesse
vitesse x
vitesse y
adaptation_ vitesse
norme_vitesse
changement_d_angle
vitesse x
vitesse y

ANNEXE 9 : Programme principal

```
287
288  # programme principal
289
290  banc = ini_banc(nombrepoisson, longueurbassin)
291  affichagepoint(banc)
292  for i in range(nombre_de_tours):
293      plt.clf()
294      tabvois = distance_entre_poisson(banc,nombre_de_voisin)
295      banc = mouvement(banc, tabvois)
296      banc = positionpoisson(banc)
297      affichagepoint(banc)
298      print(i)
299
300
301  plt.show()
```

ANNEXE 10 : Programme d'initialisation du banc de poissons

```
33  def ini_banc(nombrepoisson: int, longueurbassin: int) -> list:
34      """fonction qui permet d'initialiser le banc il prend en entree le nombre de poisson"""
35      banc = []
36      banc = [[None]*4 for i in range(nombrepoisson)]
37      for i in range(nombrepoisson):
38          vx = choice([-1, 1]) * random() * vitesse #choisi une valeur aleatoirement pour la vitesse en x
39          banc[i] = [choice(range(longueurbassin)), choice(
40              range(longueurbassin)),vx , sqrt(vitesse**2 - vx**2)*choice([-1, 1])] #adapte la vitesse en y de façon à
41                                                                                    #ce que la norme soit tout le temps égale
42          """banc[i] = [choice(range(longueurbassin//4, (3*longueurbassin)//4)), choice(
43              range(longueurbassin//4,(3*longueurbassin)//4)),vx , sqrt(vitesse**2 - vx**2)*choice([-1, 1])] #adapte la vitesse en y de fa
44      return banc
45
```

ANNEXE 11 : Programme permettant l'affichage des points représentants les poissons

```
60  def affichagepoint(banc: list):
61      """fonction qui permet d'afficher le graphique"""
62      positionx = []
63      positiony = []
64      for i in range(len(banc)):
65          positionx.append(banc[i][0])
66          positiony.append(banc[i][1])
67
68      plt.xlim(limiteaxe)
69      plt.ylim(limiteaxe)
70      plt.plot(positionx, positiony, "+b")
71      plt.gca().set_aspect('equal', adjustable='box')
72      plt.grid(True)
73      plt.pause(pause)
```

ANNEXE 12 : Programme calculant la distance entre chacun des poissons

```
134  def distance_entre_poisson(banc: list, nombre_de_voisin: int)-> list:
135      """calcul la distance entre les poissons et
136      renvoie pour chaque poisson l'indice des poissons les + proches"""
137      distancevoisin = initialisertab(nombre_de_voisin, len(banc))
138
139      for poisson in range(len(banc)):
140          compteurvoisin=0
141          indice_new_voisin=-1
142          while compteurvoisin < nombre_de_voisin:
143              indice_new_voisin+=1
144              if  indice_new_voisin!= poisson: #initialise les voisins avec les poissons initiaux
145                  distancevoisin[poisson][compteurvoisin][0]=  indice_new_voisin
146                  distancevoisin[poisson][compteurvoisin][1]= distance(positionx(banc,poisson),positiony(banc,poisson)\
147                                                                  ,positionx(banc, indice_new_voisin),positiony(banc, indice_new_voisin))
148                  compteurvoisin += 1
149      triertableau(distancevoisin)
150      for poisson in range(len(banc)):
151          for  indice_new_voisin in range(len(banc)): #modifie le tableau de voisin avec les poissons qui sont plus proche que les poissons initiaux
152              dejadedans = False
153              for x in range(nombre_de_voisin) :
154                  if distancevoisin[poisson][x][0]== indice_new_voisin :
155                      dejadedans=True
156              if poisson!= indice_new_voisin and dejadedans == False :
157                  if distance(positionx(banc,poisson),positiony(banc,poisson),positionx(banc, indice_new_voisin)\
158                              ,positiony(banc, indice_new_voisin)) < dist_voisin(poisson,nombre_de_voisin-1,distancevoisin):
159                      distancevoisin[poisson][nombre_de_voisin-1][0] = indice_new_voisin
160                      distancevoisin[poisson][nombre_de_voisin-1][1]=distance(positionx(banc,poisson),positiony(banc,poisson)\
161                                                                          ,positionx(banc, indice_new_voisin),positiony(banc, indice_new_voisin))
162                      triertableau(distancevoisin)
163      return distancevoisin
```

ANNEXE 13 : Programme codant le mouvement futur des poissons

```
273  def mouvement(banc : list, tabvois : list)->list:
274      """calcul la position finale des poissons en prenant en compte les interactions entre eux"""
275      for num_p in range(nombrepoisson):
276          if test_repulsion(banc, num_p, tabvois) == True:
277              anglerepulsion = anglefinalerepulsion(banc, num_p, tabvois)
278              banc[num_p][2], banc[num_p][3] = changement_d_angle(vitessex(banc,num_p), vitessey(banc,num_p), anglerepulsion )
279          elif test_attraction(banc, num_p, tabvois) :
280              vitesse_finale_att = attraction(banc, tabvois, num_p)
281              banc[num_p][2] = vitesse_finale_att[num_p][0]
282              banc[num_p][3] = vitesse_finale_att[num_p][1]
283          else:
284              vitesse_finale_al = alignement(banc, tabvois, num_p)
285              banc[num_p][2] = vitesse_finale_al[num_p][0]
286              banc[num_p][3] = vitesse_finale_al[num_p][1]
287      return banc
```

```
46   def positionpoisson(banc: list) -> int:
47       """fonction qui permet de calculer la prochaine position du poisson grace à sa vitesse, regarde le cas ou le
48       poisson est en dehors des limites du banc"""
49       for i in range(len(banc)):
50
51           if (banc[i][1]+ banc[i][3] < 0) or (banc[i][1]+banc[i][3] > longueurbassin):
52               banc[i][3] *= -1
53           if (banc[i][0]+banc[i][2] < 0) or (banc[i][0]+banc[i][2] > longueurbassin):
54               banc[i][2] *= -1
55           banc[i][0] += banc[i][2]*pas_temps
56           banc[i][1] += banc[i][3]*pas_temps
57       return banc
```

ANNEXE 14 : Programme renvoyant la position future des poissons

```
def positionpoisson(banc: list) -> int:
    """fonction qui permet de calculer la prochaine position du poisson grace à sa vitesse, regarde le cas ou le
    poisson est en dehors des limites du banc"""
    for i in range(len(banc)):

        if (banc[i][1]+ banc[i][3] < 0) or (banc[i][1]+banc[i][3] > longueurbassin):
            banc[i][3] *= -1
        if (banc[i][0]+banc[i][2] < 0) or (banc[i][0]+banc[i][2] > longueurbassin):
            banc[i][2] *= -1
        banc[i][0] += banc[i][2]*pas_temps
        banc[i][1] += banc[i][3]*pas_temps
    return banc
```

ANNEXE 15 : Autres fonctions générales du programme non détaillées précédemment

```python
def distance(x1,y1,x2,y2: float) -> float:
    """calcul la distance entre deux points"""
    return sqrt((x1-x2)**2 + (y1-y2)**2)

def triertableau(tabl : list) -> list:
    """tri d'un tableau de liste de liste, utilisé dans la distance au voisin"""
    erreur = True
    while erreur!=False:
        erreur = False
        for i in range(len(tabl)):
            for j in range(len(tabl[0])-1):
                if tabl[i][j][1] > tabl[i][j+1][1] :
                    switch = tabl[i][j]
                    tabl[i][j] = tabl[i][j+1]
                    tabl[i][j+1] = switch
                    erreur = True
    return tabl

def initialisertab(nombre_de_voisin, nombrepoisson : int) -> list:
    """ initialise un tableau vide qui contiendra les voisins plus tard"""
    liste= []
    for _ in range(nombrepoisson):
        sous_tableau = []
        for _ in range(nombre_de_voisin):
            soussous_tableau = [None,None]
            sous_tableau.append(soussous_tableau)
        liste.append(sous_tableau)
    return liste
```

```python
def vitessex(banc : list, numeropoisson : int)-> int:
    """permet d'obtenir la vitesse en x d un poisson"""
    return banc[numeropoisson][2]

def vitessey(banc : list, numeropoisson : int)-> int:
    """permet d'obtenir la vitesse en y d un poisson"""
    return banc[numeropoisson][3]

def positionx(banc : list, numeropoisson : int)-> int:
    """permet d'obtenir la position en x d un poisson"""
    return banc[numeropoisson][0]

def positiony(banc : list, numeropoisson : int)-> int:
    """permet d'obtenir la position en y d un poisson"""
    return banc[numeropoisson][1]

def norme_vitesse(vx , vy)-> int:
    """calcul de la norme d'une vitesse"""
    return sqrt(vx**2 + vy**2)

def num_voisin(numero_poisson: int, numerovoisin:int, distancevoisin : list)-> int:
    """permet d'obtenir le numéro d'un voisin d'un poisson donné
    exemple : le numéro du premier voisin du poisson 4 est ..., num_voisin(4,0,tab_voisin)-> ... """
    return distancevoisin[numero_poisson][numerovoisin][0]

def dist_voisin(numero_poisson: int, numerovoisin : int, distancevoisin : list)-> int:
    """permet d'obtenir la distance d'un voisin d'un poisson donné avec lui même
    exemple : la distance du premier voisin du poisson 4 avec lui même est ..., dist_voisin(4,0,tab_voisin)-> ... """
    return distancevoisin[numero_poisson][numerovoisin][1]

def changement_d_angle(vitesseinix : int, vitesseiniy : int, angle : int) -> list:
    """permet de calculer la vitesse en x et en y d'un poisson en connaissant l'angle qu'il doit effectuer"""
    anglefinal = atan2(vitesseiniy,vitesseinix)-angle*pi/180
    normevitesse = norme_vitesse(vitesseinix, vitesseiniy)
    vitessefinaley = sin(anglefinal)*normevitesse
    vitessefinalex = cos(anglefinal)*normevitesse
    vitessefinale = [vitessefinalex,vitessefinaley]
    return vitessefinale
```