REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre:35032599/M2/2021

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Image et Vie Artificielle (IVA)**

Optimisation de mouvement coordonné de robots en essaim avec les algorithmes génétiques.

Par :

BOUHADOU LEILA

Soutenu le ../../…. devant le jury composé de :

Akrour Djaoher	grade	Président
Nom Prénom	grade	Rapporteur
Nom Prénom	grade	Examinateur

Année universitaire 2020-2021

Remerciements

Je tiens à exprimer toute ma reconnaissance à ma directrice de mémoire Akrour Djaouher. Je la remercie de m'avoir encadré, orienté, aidé et conseillé.

Je remercie mes très chers parents pour tous leurs sacrifices, leurs amours, leurs soutiens et leurs prières tout au long de mes études,

Je remercie mon cher mari et mes chers frères pour leurs encouragements.

Je tiens à remercier mon chère amie Iktam Maouche pour leur soutien et leur aide.

Résumé

La robotique en essaim est un domaine émergent de la robotique, qui s'intéresse aux systèmes multi-robots auto-organisés, souvent caractérisés par de grands nombres d'unités. L'un des challenges de la robotique en essaim est de pouvoir se déplacer de manière coordonnée tout en réalisant des tâches collectives. Ce comportement doit être émergé de manière automatique sans qu'il y ait un leader qui dicte le rôle de chacun des robots. Cela permet à l'essaim d'être plus effectifs lors de la réalisation de tâches complexes.On utilise pour cela les algorithmes issus de l'intelligence en essaim inspirés de la nature et des comportements des animaux tels que les volées d'oiseaux (*Flocking*). Notre travail consiste à concevoir un comportement coordonné de robots simulés en 2D en utilisant un algorithme de *Flockage* (Boids) et un algorithme évolutionnaire (AG) pour l'optimisation du comportement obtenu.

TABLE DES MATIÈRES

Sommaire

Liste des figures:

Liste des Tableaux :

Introduction générale

La *Swarm Intelligence* ou intelligence en essaim est une discipline bio-inspirée qui s'intéresse particulièrement à l'intelligence qui émerge à partir de comportements collectifs chez certaines espèces, vivant dans des groupes et possèdent des propriétés intéressantes, et essaye d'adapter ces propriétés aux problèmes réels.

Dans le domaine de l'informatique, depuis ses débuts, il a toujours semblé intéressant de s'inspirer de phénomènes, de comportements sociaux ou collectifs chez certaines espèces vivantes. Malgré la richesse des idées dans plusieurs disciplines telles que la zoologie, les moyens de calculs de l'époque antérieure aux années 90 étaient vraiment insuffisants et les problèmes qui surgissaient devenaient vite insurmontable.

De nos jours, précisément durant cette dernière décennie, et avec l'apparition de plusieurs défis (masses gigantesques de données, système distribués, modélisation et extraction de connaissance), plusieurs chercheurs en informatique ont orienté leurs recherches vers des approches très intéressantes qui s'inspirent directement de la nature. C'est le cas, par exemple, des insectes sociaux. Ces derniers arrivent à survivre depuis des millions d'années dans des conditions très difficiles et un environnement imprévisible. Ces insectes possèdent des caractéristiques remarquables telles que l'autonomie, l'auto-organisation, l'optimisation, le contrôle décentralisé…, ce qui a attiré beaucoup de chercheurs vers la voie bio-inspirée afin de résoudre des problèmes complexes tels que ceux de l'optimisation.

L'idée principale derrière la robotique en essaim consiste à étudier comment coordonner de grands groupes de robots relativement simples grâce à l'utilisation de règles locales. Il se concentre sur l'étude de la conception d'un grand nombre de robots relativement simples, de leur corps physique et de leurs comportements de contrôle afin de réaliser une tâche spécifique qui dépasse les capacités d'un seul robot. La robotique en essaim est étroitement liée à l'idée d'intelligence en essaim et partage son intérêt pour les systèmes décentralisés auto-organisés. Il offre plusieurs avantages pour les applications robotiques telles que l'évolutivité, la flexibilité et la robustesse dues à la redondance. avec les énormes progrès réalisés dans ce domaine, les

chercheurs se sont principalement concentrés sur la manière dont un système de robot essaim peut être impliqué dans notre vie réelle.

Le but de notre travail consiste donc a concevoir un comportement coordonnés de robots simulés en 2D en utilisant un algorithme de *Flockage* et un algorihtme évolutionnaire pour l'optimisation du comportement obtenu.

Le mémoire est organisé en 3 chapitres regroupés en deux parties principales : La première partie (chapitre 1 et chapitre 2), il place le lecteur dans le contexte du travail et lui présente progressivement l'état de l'art et les études connexes de la recherche présentée dans ce mémoire, dans la seconde partie (chapitre 3) de cette partie, nous avons présenté notre approche validée par quelques résultats de simulation.

Au chapitre 1, un aperçu détaillé du domaine de la robotique en essaim est présenté de manière chronologique. Le chapitre commence par présenter le domaine de l'intelligence des essaims et ses principaux concepts fondamentaux. Il introduit ensuite le domaine de la robotique en essaim en tant que sous-domaine spécifique à la multi-robotique dans lequel la théorie de l'intelligence en essaim est appliquée, en précisant les principales différences entre les deux domaines de recherche (biologie, robotique).

Le chapitre 2, nous allons présenter un état de l'art de certain travaux présentés dans la littérature, travaillant sur la conception de mouvement coordonnées appliqués au robots en essaim. Ensuite, nous définissons les deux algorithmes que nous allons utiliser dans notre propre travail, l'algorithme Boids de flocage et les algorithmes génétiques.

Le chapitre 3, ce chapitre se compose de deux phases : la conception et l'implémentation de notre projet.

Chapitre I

L'intelligence et la robotique en essaim

Chapitre 1: Intelligence et robotique en essaim

1.1 Introduction

La robotique en essaim est une branche de la robotique appliquant les méthodes d'intelligence distribuée aux systèmes à plusieurs robots. Il s'agit généralement d'utiliser des robots simples, voire simplistes, et peu coûteux, d'un intérêt individuel assez limité, mais qui ensemble (par exemple via des capacités d'auto-assemblage ou d'auto-organisation) forment un système complexe et robuste.

L'une des méthodes de l'intelligence distribuée est la *Swarm Intelligence* ou intelligence en essaim. C'est une discipline bio-inspirée qui s'intéresse particulièrement à l'intelligence qui émerge à partir de comportement collectifs chez certaines espèces, vivant dans des groupes et possèdent des propriétés intéressantes, et essaye d'adapter ces propriétés aux problèmes réels.

Dans ce chapitre nous allons décrire en détail cette forme d'intelligence et son application dans la robotique en essaim. Nous allons également présenter une classification des types de problèmes qu'on peut traiter dans ce domaine de la robotique.

1.2 L'intelligence en essaim

1. 1. 2.1. Intelligence collective dans la nature

Dans la nature et presque chez tous les insectes ou les animaux qui vivent ensemble dans des groupes, on remarque une sorte d'intelligence qui émerge à partir du comportement collectif et social. Le niveau d'intelligence varie selon l'espèce (fourmis, abeilles, oiseaux, poissons…) et l'environnement et le niveau d'interaction et d'autres facteurs.

Nous essayons dans ce qui suit d'éclairer un peu les idées concernant la swarm

intelligence par l'illustration de quelque exemple.

Figure 1.1 : Exemples de systèmes d'essaims naturels.

1.1.2. L'intelligence en essaim

L'intelligence en essaim (ou Swarm Intelligence: SI) est une discipline d'intelligence artificielle moderne qui est concernée, entre autres, par la conception de systèmes multi-agents. Le paradigme de conception pour ces systèmes est fondamentalement différent des approches plus traditionnelles. [1]

Ainsi, l'intelligence en essaim est une voie informatique innovatrice pour la résolution des problèmes difficiles. En général, cette résolution est inspirée du comportement des créatures biologiques dans leurs essaims et colonies. [2]

Elle est considérée comme une métaphore comportementale pour résoudre des problèmes

distribués inspirés d'exemples biologiques fournis par des insectes sociaux comme les fourmis, les termites, les abeilles ou les guêpes et par essaim, troupeau, masse et phénomènes de bancs chez les animaux comme les bancs de poissons et les masses d'oiseaux. [3]

2. 2.3. Caractéristiques

La *Swarm intelligence* a les caractéristiques notables suivantes: [3]

- *Autonomie*: Le système n'exige pas de gestion extérieure ou de maintenance, les individus sont autonomes, contrôlant leur propre comportement d'une manière auto-organisée.
- *Adaptabilité*: Les interactions entre des individus peuvent surgir par la communication directe ou indirecte via l'environnement local. Deux individus interagissent indirectement quand l'un d'entre eux modifie l'environnement et l'autre répond au nouvel environnement à un temps postérieur. En exploitant de telles formes de communication locales, les individus ont la capacité de détecter des changements au sein de l'environnement dynamiquement, ils peuvent alors adapter leur propre comportement à ces nouveaux changements d'une manière autonome. Ainsi, les systèmes d'essaims présentent des capacités d'auto-configuration.
- *Evolutivité*: Les capacités de la *swarm intelligence* peuvent être optimisées en utilisant des groupes comportant un nombre variant de quelques individus jusqu'à des milliers avec la même architecture de contrôle.
- *Flexibilité*: Aucun individu simple de l'essaim n'est essentiel, c'est-à-dire n'importe quel individu peut être dynamiquement ajouté, enlevé ou remplacé.
- *Robustesse*: La *swarm intelligence* fournit un bon exemple d'architecture fortement distribuée, ce qui améliore considérablement sa robustesse. Aucune coordination centrale n'a lieu d'être, ce qui signifie qu'il n'y a aucun point seul d'échec. De plus, comme la plupart des systèmes biologiques et sociaux, et en combinant les capacités d'adaptabilité et flexibilité, le système d'essaim permet la redondance qui est essentielle pour la robustesse.
- *Parallélisme massif*: Le système d'essaim est massivement parallèle et son fonctionnement est vraiment distribué. Les tâches exécutées par chaque individu

dans son groupe sont les mêmes.

- *Rentabilité*: Les systèmes d'essaim consistent en une collection finie d'agents homogènes, chacun d'entre eux a les mêmes capacités et les mêmes algorithmes de contrôle. Il est clair que l'autonomie et le contrôle fortement distribué, permis par le modèle en essaim, simplifient considérablement la tâche de mise en œuvre d'algorithmes parallèle.
- *Auto-organisation:* Les systèmes d'essaim soulignent des capacités d'auto-organisation. L'intelligence n'est pas présente au niveau de chaque individu, mais apparaît plutôt d'une façon ou d'une autre dans l'essaim entier. Autrement dit, si nous considérons chaque individu comme une unité de traitement, les solutions aux problèmes obtenus ne sont pas prédéterminées ou préprogrammées, mais sont déterminées collectivement suite au programme d'exécution. Par exemple, pour le type d'essaim des systèmes multi-robotisés, les robots sont relativement simples et leur effort de processus de conception peut être maintenu minimal en termes de capteurs, de déclencheurs et de ressources pour le calcul et la communication.

1. 2.4. Applications de la swarm intelligence

Les principes de l'intelligence en essaim ont été appliqués avec succès dans une grande variété de domaines, bien que l'optimisation par colonie de fourmis et l'optimisation par essaim de particules représentent une direction de recherche réussie dans le domaine d'intelligence en essaim, d'autres exemples d'application intéressants peuvent être considérés, tel que la Gestion des réseaux à base d'essaims et le comportement coopératif dans les essaims de robots [3].

1. 2.5. Avantages de la swarm intelligence

L'intelligence en essaim fournit une nouvelle structure pour la conception et la mise en œuvre de systèmes composés de plusieurs agents qui sont capables de coopérer pour résoudre un problème complexe. Les avantages potentiels de l'intelligence en essaim sont variés [4]:

- ✓ La robustesse collective: L'échec de composants individuels ne gêne pas significativement la performance.
- ✓ Le comportement coopératif et la simplicité individuelle: Permettent de réduire la complexité des individus.

- ✓ L'adaptabilité: Les mécanismes de contrôle utilisés ne dépendent pas du nombre d'agents dans l'essaim.

L'intelligence en essaim peut donc être vue comme une ingénierie pratique. La majorité des recherches s'est concentrée, jusqu'à présent, sur des démonstrations concernant les capacités des systèmes intelligents à base d'essaim pour la résolution de problèmes nécessitant une coopération. Des résultats très encourageants ont été obtenus, particulièrement dans des applications d'optimisation [4].

Les essaims fournissent une source d'inspiration riche et leurs principes sont directement applicables aux systèmes informatiques. Cependant, bien qu'elle se caractérise par l'auto configuration, l'auto-organisation et la capacité d'adaptation, l'approche en essaim reste utile pour des applications impliquant de nombreuses répétitions de la même activité sur un secteur relativement grand, comme la découverte du chemin le plus court.

En effet, l'approche de type "essaim" est basée sur la coopération de grands nombres d'agents homogènes. De telles approches comptent d'habitude sur des résultats de convergence mathématiques qui atteignent le résultat désirable au cours d'une période de temps suffisamment longue. De plus, les agents impliqués sont homogènes [3].

1. 2.6. L'auto-organisation

L’auto-organisation est un phénomène très intéressant par lequel un système peut avoir la capacité de s'organiser lui-même. C’est une caractéristique très importante que nous cherchons à appliquer dans le domaine de la robotique.

On remarque ce phénomène dans les systèmes physiques, biologiques, écologiques ou sociaux. Ceux-là ont tendance à s'organiser d'eux-mêmes. Soit initialement lors de leurs émergence spontanée, ou soit lorsque le système existe déjà et auto-organisation s’applique pour une organisation plus structurée ou complexe.

L'auto-organisation se produit par des interactions internes et externes au système, au sein de son milieu et avec lui. Elle consomme de l'énergie qui sert ainsi à établir et maintenir le système auto-organisé. L'auto-organisation s'oppose aux cas où un système est organisé ou réorganisé de force de l'extérieur, c'est-à-dire à la violence, aux actes de pouvoir : cela rejoint

aussi le contraste entre autonomie et hétéronomie.

Typiquement, un système auto-organisé a des propriétés émergentes. Passé un seuil critique de complexité, les systèmes peuvent aussi changer d'état, ou passer d'une phase instable à une phase stable ou inversement.

1. 2.6.1. Caractéristiques de l'auto-organisation biologique

Les principales propriétés des phénomènes biologiques d'auto-organisation sont les suivants [5] :

– Ce sont des processus dynamiques, dont les éléments constitutifs sont en interaction permanente pour maintenir l'auto-organisation du système. Ces processus dynamiques sont de nature non linéaire.

– Ces processus donnent lieu à des propriétés nouvelles émergentes, non présentes au sein des parties élémentaires et non simplement liées à leurs propriétés élémentaires (formation d'amas chez certaines larves d'insectes par exemple). De plus ces processus peuvent mettent en œuvre des mécanismes localement simples mais qui produisent `a grande échelle des résultats complexes.

– Ces processus sont paramétriques: certains paramètres liés au fonctionnement biologique (la composition chimique d'une phéromone par exemple) et d'autres de nature physique (par exemple la volatilité d'une phéromone, la température, les mouvements de l'air vont jouer sur son évaporation) peuvent avoir un fort impact sur les processus d'auto-organisation. Cette dépendance paramétrique renforce aussi le rôle et l'importance de l'environnement. De fait, certaines de ces propriétés rejoignent celles déjà constatées dans d'autres domaines (en physique notamment).

1. 3. Robotique en essaim

La Robotique en essaim selon Dorigo, Birattari et Brambilla [6] est l'étude de la manière de créer des groupes de robots qui s'exécutent sans dépendre d'aucune infrastructure extérieure ou aucune forme de contrôle centralisée. Dans un essaim de robots, le comportement collectif des robots résulte de l'interaction locale entre les robots et l'environnement dans lequel ils interagissent. Les systèmes de robotique en essaim sont des systèmes tolérants aux erreurs,

extensible et flexible.

Ce sont des systèmes multi-robots décentralisés avec une communication explicite et une coordination dynamique souvent copiée sur la nature. Ils permettent avec un nombre d'unités simple d'effectuer des comportements collectifs complexes comme le *flocking* (ou vol en nuée en français).

C'est une approche prometteuse lorsque différentes activités doivent être effectuées simultanément, lorsque la redondance élevée et l'absence d'un point de défaillance unique sont souhaitées, et quand il est techniquement impossible de mettre en place l'infrastructure nécessaire pour contrôler les robots de façon centralisée.

1. 3.1. Caractéristiques de la robotique en essaim

À la différence de la plupart des systèmes robotiques répartis, la robotique en essaim insiste sur un grand nombre de robots et promeut la mise à l'échelle, par exemple l'utilisation de communications locales sous forme d'infrarouge ou Sans-fil. On attend de ces systèmes qu'ils possèdent au moins les trois

propriétés suivantes :

- La robustesse, qui implique la capacité de l'essaim à continuer à fonctionner malgré les défaillances de certains individus le composant et/ou les changements qui peuvent survenir dans l'environnement ;
- La flexibilité, qui implique une capacité à proposer des solutions adaptées aux tâches à réaliser ;
- La « mise à l'échelle », qui implique que l'essaim doit fonctionner quelle que soit sa taille (à partir d'une certaine taille minimum).

Selon [4] dans un système robotique en essaim :

- ✓ Chaque robot est autonome ;
- ✓ les robots sont habituellement capables de se situer par rapport à leurs voisins les plus proches (positionnement relatif) et parfois dans l'environnement global, même si certains systèmes essayent de se passer de cette donnée ;
- ✓ Les robots peuvent agir (ex : pour modifier l'environnement, coopérer avec un autre

robot.);

- ✓ Les capacités de détection et de communication des robots entre eux sont locales (latérales) et limitées ;
- ✓ Les robots ne sont pas reliés à un contrôle centralisé ; ils n'ont pas la connaissance
- ✓ globale du système dans lequel ils coopèrent ;
- ✓ Les robots coopèrent pour effectuer une tâche donnée ;

1. 3.2. Avantages et Limites

Les avantages les plus souvent cités sont :

- Un faible coût pour une couverture plus étendu une capacité de redondance (si l'un des robot est défaillant en raison d'une panne, d'un blocage, etc. un autre robot peut prendre des mesures pour le dépanner ou le remplacer dans sa tâche).

- La capacité à couvrir une grande surface. Ils ont par exemple montré (via une simulation appliquée au cas de l'île de Lampedusa) en 2014 qu'un essaim de 1000 petits drones aquatiques dispersés en mer à partir de 2 bases pourraient en 24 heures faire un bilan de surveillance sur une bande maritime longue de 20 km.

À ce jour, les essaims de robots ne peuvent remplir que des taches relativement simples, ils sont souvent limités par leur besoin en énergie. De manière plus générale, les difficultés d'interopérabilité quand on veut associer des robots de nature et d'origines différentes sont aussi encore très limitantes.

1. 3.3. Domaine d'application

Les applications potentielles de la robotique en essaim comprennent les tâches qui exigent la miniaturisation, telles que les tâches de détection distribuées dans des micromachines ou le corps humain. D'autre part, la robotique en essaim peut être adaptée aux tâches qui requièrent des conceptions bon marché, telles que la tâche d'extraction ou la tâche de recherche de nourriture .

La robotique en essaim peut également être impliquée dans des tâches qui nécessitent un coût en espace et en temps important, et sont dangereuses pour l'homme ou les robots eux-mêmes, telles que l'aide post-sinistre, la recherche de cible, les applications militaires, etc.

Plusieurs domaines d'applications potentiels pour les systèmes de robotique en essaim qui sont très appropriés sont décrits ci-dessous, classés principalement en quatre types:

1. 3.3.1. Les tâches couvrent une grande surface

Le système de robotique en essaim est distribué et spécialisé pour les tâches nécessitant une grande surface d'espace, par exemple, les tâches couvrent de grandes surfaces. Un exemple simple serait la recherche et la collecte de plusieurs cibles dans une zone ouverte. L'essaim essaie de rechercher avec la coopération de groupe pour accélérer la recherche. La zone peut être très vaste et l'essaim peut tirer parti de la recherche parallèle avec plusieurs petits groupes situés dans les zones de détection des robots.

1. 3.3.2. Tâches dangereuses pour le robot

Grâce à l'évolutivité et à la stabilité, l'essaim fournit une redondance pour traiter des tâches dangereuses. L'essaim peut subir une perte considérable de robots avant que le travail ne soit interrompu. Les robots sont très bon marché et sont préférés pour les zones qui endommagent probablement les travailleurs. Dans certaines tâches, les robots peuvent être irrécupérables et l'utilisation de robots complexes et coûteux est donc inacceptable sur le plan économique, tandis que la robotique en essaim avec des individus bon marché peut fournir des solutions raisonnables.

1. 3.3.3. Les tâches nécessitent une population d'échelle

La charge de travail de certaines tâches peut changer avec le temps, et la taille de l'essaim doit être réduite en fonction de la charge de travail actuelle pour une efficacité élevée, à la fois en termes de temps et d'économie. Par exemple, pour éliminer les fuites d'huile après les accidents de la citerne, l'essaim devrait maintenir une population élevée lorsque l'huile fuit rapidement au début de la tâche et réduire progressivement le nombre de robots lorsque la source de la fuite est bouchée et que la zone de fuite est presque entièrement nettoyée.

1. 3.3.4. Les tâches nécessitent une redondance

La robustesse des systèmes robotiques en essaim profite principalement de la redondance de l'essaim, c'est-à-dire que le retrait de certains robots n'a pas d'impact significatif sur les performances. Certaines tâches se concentrent sur le résultat plutôt que sur le processus, c'est à-dire que le système doit s'assurer que la tâche sera complétée avec succès, principalement sous la

forme d’une redondance croissante.

1. 3. 4. Les problèmes traités en robotique en essaim

Dans le tableau ci-dessus, nous présentons l’étude faite par Brambilla et al. [7] dans laquelle les auteurs donnent une brève définition du problème à résoudre, sa source d’inspiration, les approches utilisées pour modéliser le problème, des exemples des recherches en cours qui appartiennent au problème et enfin la classification des problèmes.

Problématique	**Sources d'inspiration**	**Approches de modélisation**
Agrégation: regroupement de robots d'essaims dans une région de l'environnement.	Nature (par exemple, bactéries d'agrégation, blattes, abeilles, poissons, etc.).	• Machines à états finis probabilistes. • Évolution artificielle
Formation de motifs: déployer des robots de manière régulière et répétitive en respectant des distances spécifiques afin de créer le motif souhaité.	• Biologie (par exemple, la disposition spatiale des colonies bactériennes et les schémas chromatiques de certains animaux). • Physique (distribution de molécules et formation de cristaux).	Conception virtuelle basée sur la physique.

Formation de chaînes: des robots à positionnement automatique se connectent en deux points. La chaîne qu'ils forment peut ensuite être utilisée comme guide de navigation ou de surveillance.	Fourmis	• Machines à états finis probabilistes. Conception basée sur la physique virtuelle. Évolution artificielle
Auto-assemblage et morphogenèse: Relier des robots d'essaims physiques pour créer des structures (morphologies).	Fourmis (ponts, radeaux, murs…).	• Machines à états finis probabilistes. Conception basée sur la physique virtuelle. • Évolution artificielle
Exploration collective	Animaux sociaux (fourmis, abeilles, etc.).	• Machines à états finis probabilistes. • Conception virtuelle basée sur la physique. • Routage réseau.
Transport collectif: Coopérer pour transporter un objet.	Les coopératives portent leurs proies dans les colonies de fourmis.	• Machines a état finis probabiliste. Evolution artificielle.
Comportement coordonné : Se déplaçant en formation de la même manière que les bancs de poissons ou les volées d'oiseaux.	• Flocage en groupe d'oiseaux. • Scolarisation en groupe de poissons.	• Conception basée sur la physique virtuelle. • Évolution artificielle

Tableau 1.1 *:* Problèmes de la robotique en essaim

Dans notre travail, nous nous intéressons à la conception de comportements coordonnés de robots en essaims en utilisant l'algorithme de Boids de Reynolds et un apprentissage par les algorithmes génétiques.

1. 4. Conception de comportement émergent

1. 4.1. Propriétés du comportement collectif

L'organisation des populations en essaim est caractérisée par un phénomène de groupes, dont les particularités sont les suivantes:

- Information locale limitée: Chaque individu possède une connaissance locale de l'environnement et ne connaît rien de la fonction globale de la population.
- Des règles individuelles simples: Chaque individu possède un jeu de règles comportementales simples limitées. Ce jeu de règles permet au groupe de coordonner ses actions collectivement et de construire une structure ou d'adopter une configuration globale.
- La structure globale émergente accomplit une certaine fonction: Ces structures permettent au groupe de résoudre un problème. Pour cela, ces structures sont
- flexibles (adaptatives à l'environnement) et robustes.

1. 4.2 Comportement émergent

Est le comportement d'un système qui ne dépend pas de ses parties individuelles, mais de leurs relations les unes avec les autres. Ainsi, un comportement émergent ne peut pas être prédit par l'examen des parties individuelles d'un système.

Le comportement émergent a également été défini comme l'action de règles simples se combinant pour produire résultats complexes.

Le comportement émergent est l'action collective d'un groupe d'agent, non implémentée de façon explicite, conséquence d'actions individuelles non prédictives.

1. 5. Conclusion

La robotique en essaim est un domaine de recherche relativement nouveau qui s’inspire de l’intelligence en essaim. C'est le résultat de l'application de techniques d'intelligence en essaim à la multi-robotique. Dans ce chapitre, nous avons présenté un aperçu de la robotique en essaim ainsi que les théories basées sur ce domaine.

Dans le chapitre suivant nous décrirons les méthodes et les algorithmes que nous allons appliquer pour la conception du comportement coordonnés de robots en essaims mobiles.

Chapitre 2

Conception de mouvement coordonné et optimisé

2.1 Introduction

L'un des challenges de la robotique en essaim est de pouvoir se déplacer de manière coordonnée tout en réalisant des tâches collectives (navigation, exploration, transport, etc.). Ce comportement doit être émergé de manière automatique sans qu'il y ait un leader qui dicte le rôle de chacun des robots. On utilise pour cela les algorithmes issus de l'intelligence en essaim inspirés de la nature et des comportements des animaux tels que les volées d'oiseaux (Flocking). Dans ce chapitre, nous allons présenter un état de l'art de certain travaux cités dans la littérature, travaillant sur la conception de mouvements coordonnées appliqués au robots en essaim. Ensuite, nous définissons les deux algorithmes que nous allons utiliser dans notre propre travail, l'algorithme Boids de flocage et les algorithmes génétiques.

2.2 Mouvement collectif

Concevoir un mouvement collectif consiste à étudier la problématique de coordonner un groupe de robots et de les déplacer ensemble en tant que groupe d'une manière cohérente . Il peut également servir de comportement de base pour réaliser des tâches plus compliquées, par exemple l'exploration, le transport et la navigation.

Le mouvement collectif peut être classé en deux types: formations et flocage. Dans le premier cas, les robots doivent maintenir des positions et des orientations prédéterminées entre eux. En revanche, dans le flocage, les positions relatives des robots ne sont pas strictement.

2.3 Coordination des robots en essaim

On a proposé dans la littérature de nombreuses architectures de conception de mouvements collectifs. Cependant, seules celles permettant une évolutivité avec un nombre croissant de robots sont intéressantes. Pour cela, on a utilisé des lois de contrôle inspirées de la Physique. Ici, le contrôleur est entièrement décentralisé; chaque robot perçoit les positions relatives de ses voisins et réagit aux forces attractives ou répulsives, formant des treillis triangulaires. L'algorithme est évolutif et fonctionne pour des dizaines de robots.

Lee et Nak [11] proposent un algorithme distribué de mouvement collectif basé sur des

treillis. Sa convergence est prouvée en utilisant le théorème de Lyapunov.

Dans [12]un algorithme décentralisé pour le mouvement collectif basé sur des formations en treillis est proposé. La stabilité de l'algorithme dans un cas d'étude particulier est prouvée. L'évitement d'obstacles est mis en œuvre en partitionnant le plan en régions de Voronoi.

Turgut et coll. [13] proposent et étudient un algorithme évolutif et distribué pour le flocage de robots. Il est basé sur l'alignement de cap des robots et le contrôle de distance inter-robot. L'algorithme est testé en simulation avec jusqu'à 1000 robots et avec un petit groupe de vrais robots.

2. 3.1 Challenges rencontrés

Afin de maintenir un comportement coordonné, chaque robot de l'essaim doit avoir la capacité de coordonner et partager la charge de travail avec l'autre. Cependant, certains problèmes surviennent toujours dans la coordination, comme les tâches allocation pour le groupe de robots, y compris: l'utilisation des ressources; accomplissement des tâches en temps; excessif communication, sélections de capteurs, fiabilité du système et évolutivité. [11]

Certains chercheurs ont tenté de surmonter les problèmes en apportant quelques améliorations [12–16]. Dans [12] les auteurs ont proposé ce qui peut réduire les temps et les ressources pendant le processus de coordination.

Il peut prendre en charge l'essaim pour partager les données des capteurs et suivre ses membres. Pour améliorer la durée de vie des réseaux, A. Wichmann [14] a établi un capteur pour la communication et la coordination du robot [14] qui peut réduire la consommation d'énergie.

Corkeet.al [15] a également analysé les robots qui ont travaillé ensemble en utilisant un réseau de capteurs. M. Schwager dans [16] a utilisé le réseau de capteurs de certains nœuds. Ces capteurs ont la capacité de détecter la valeur de la fonction sensorielle élevée d'une zone. Il détectera l'observation dans une densité plus élevée.

2. 4. Méthodes de contrôle utilisées

2. 4.1. Manuelle

Dans cette catégorie de conception de contrôle ou de mouvement coordonné, on utilise généralement des méthodes conçues manuellement ou des modèles mathématiques. Cependant, l'algorithme le plus utilisé est l'algorithme *Boids* de Reynolds. Il permet de simuler le comportement de flocage des oiseaux en utilisant trois règles basiques.

2. 4.1.1. Algorithme de Boids

Le travail de Reynolds sur la synthèse du *flocking* ou de flocage est une des premières et plus remarquables propositions de règles permettant de simuler un groupe d'entités virtuelles informatiques se déplaçant collectivement en évitant des obstacles. Bien que peu formalisé, l'article [8] apporte un ensemble de spécifications précises tout en laissant le choix de la méthode d'implémentation.

- ***L'objectif***

L'objectif premier de Reynolds est la simulation et la performance algorithmique, non l'explication des phénomènes décisionnels internes aux individus du groupe. L'exigence de réalisme pour la simulation donne au projet son caractère bio-inspiré : les tentatives d'évaluation par rapport à des données expérimentales restent peu nombreuses (comparaison statistique, schémas de formation,. . .), mais les simulations obtenues ont pu malgré tout convaincre les biologistes. En réalité ce sont les systèmes particulaires déjà existants en infographie (notamment les modèles de Reeves [9]) —avec lesquels les simulations de fumées, vagues, nuages sont réalisables— qui sont la source d'inspiration algorithmique des Boids de Reynolds.

L'apport de l'algorithme de Reynolds à ce moment-là tient dans sa rupture avec l'approche de modélisation consistant à calculer individuellement les trajectoires des membres du groupe de façon globale. Reynolds propose en effet un calcul local d'interactions basées sur des perceptions locales. Ces mécanismes ont pour effet de voir émerger la formation des groupes de façon spontanée au niveau global du système. Reynolds parle alors d'acteurs, concept précurseur de nos agents actuels (réactifs en l'occurrence).

Figure 2.1 : Les trois règles fondamentales de flocage

- **Règles de conception**.

Les trois règles fondamentales édictées par Reynolds à la base des décisions des "Boids" consistent en [8] :

– **la séparation** : éviter les collisions avec les plus proches voisins, ce qui revient à respecter une distance minimale avec les autres,

– l'alignement : adapter sa vitesse à celle de ses voisins, et rester dans la direction commune de déplacement,

– la cohésion : rester proche de ses voisins en se rapprochant du centre de la nuée.

Figure 2.2 : règle de déplacement et d'évitement d'obstacle

Quelques règles ou précisions complémentaires viennent ajuster l'application de ces trois règles de base :

- Chacune de ces règles induit une accélération de l'agent (c'est-`a-dire la médication de son vecteur vitesse; (cf. figure 2.a)). Un facteur d'importance relative est associé à chacune de ces accélérations. En règle générale, on fait la moyenne des trois, mais si une action devient plus urgente que les autres, par exemple se séparer parce que deux agents sont trop proches, alors on renforce le poids de la règle qui corrige l'évènement perçu. On peut également définir un ordre de priorité sur les règles et une limite à la norme des accélérations pour chaque agent.

- La règle d'alignement ne prend pas en compte la position des agents, tandis que les deux autres règles ont besoin de cette information.

- D'autre part, la règle de cohésion ne s'applique banalement qu'au centre du voisinage de perception et non au centre de la nuée globale. Il s'agit d'un calcul barycentrique sur les agents dans le voisinage de perception. Cette règle modifier peu le déplacement d'agents dans la nuée, mais ramène ceux du bord vers le centre.

- Cette même règle autorise la division du groupe en plusieurs parties, notamment pour gérer le passage d'un obstacle, contrairement aux modèles à forces centrales ou à agent "leader". En revanche la localité associée à l'application de cette règle rend la réunion de deux parties distinctes délicate puisque la perception limitée des agents ne leur permet pas forcément de réunir le groupe.

En termes d'implémentation, chaque règle utilise des modélisations ou paramètres septiques :

– La perception limitée est implémentée par une sphère de rayon limité et une sensibilité inversement proportionnelle à l'exponentielle de la distance. Cette perception est en général réduite à un cône pour favoriser la perception dans le sens du mouvement.

– La séparation et la cohésion utilisent des forces d'attraction et de répulsion inversement proportionnelles au carré de la distance.

- **L'´évitement d'obstacles.**

L'évitement d'obstacles est un comportement de base présent dans quasiment tous les robots mobiles. Il est indispensable pour permettre au robot mobile de fonctionner dans un environnement inconnu et pour gérer les écarts entre le modèle interne et le modèle réel.

Deux approches principales sont développées pour l'évitement d'obstacles : [9] [10]

- ✓ ***Une approche à base de champs de forces*** *:* un champ de force répulsive est présent autour de l'obstacle. Mais cette technique rencontre plusieurs d´défauts : certaines configurations d'incidence de l'agent sur l'obstacle ne produisent aucune modification (ex. : longer l'obstacle, ou l'aborder exactement de front), de plus la vision périphérique des agents peut produire des mouvements inattendus (ex. : se retourner sur l'obstacle).

- ✓ ***Une approche "guidage-pour-éviter"*** donnant de meilleurs résultats de simulation, consiste pour l'agent à se laisser guider par l'obstacle. Plusieurs variantes existent, en voici deux des plus efficaces :

– L'agent utilise un point test situé en avant dans sa direction de déplacement —c'est le point où il se situerait quelques pas de temps plus tard si aucun changement de direction ne se produisait (cf. figure 2.b) — à partir duquel il calcule un nouvel angle de déplacement si ce point entre en contact avec l'obstacle. Cette version simple est efficace pour un coût computationnel réduit.
L'agent suit le bord de l'obstacle en estimant la silhouette de l'arête la plus proche de l'obstacle. Cela implique pour l'agent d'avoir accès `a une information sur la forme de l'obstacle qui lui permette de calculer sa silhouette par projection. Cette seconde version est plus lourde à mettre en œuvre. [10]

2.4.2. Automatique

Dans cette catégorie de conception de mouvement coordonné, on utilise les méthodes d'apprentissage automatique, plus spécialement les algorithmes évolutionnaire tel que les algorithmes génétiques. Ils servent à optimiser des comportements adaptables à l'environnement de simulation.

2.4.2.1. Les algorithmes génétiques

Comme dans les systèmes biologiques soumis à des contraintes, les meilleurs individus

de la population sont ceux qui ont une meilleure chance de se reproduire et de transmettre une partie de leur héritage génétique à la prochaine génération. Une nouvelle population, ou génération, est alors crée en combinant les gènes des parents. On s'attend à ce que certains individus de la nouvelle génération possèdent les meilleures caractéristiques de leurs deux parents, et donc qu'ils seront meilleurs et seront une meilleure solution au problème.

Le nouveau groupe (la nouvelle génération) est alors soumis aux même critères de sélection, et par après génère ses propres rejetons. Ce processus est répété plusieurs fois jusqu'à ce que tous les individus possèdent le même héritage génétique. Les membres de cette dernière génération, qui sont habituellement très différents de leurs ancêtres, possèdent de l'information génétique qui correspond à la meilleure solution au problème développé.

- **Principe d'un algorithme génétique**

Les algorithmes génétiques sont des algorithmes d'optimisation s'appuyant sur des techniques dérivées de la génétique et de l'évolution naturelle, tels que les croisements, la mutation et la sélection. Un algorithme génétique recherche le ou les extrema d'une fonction définie sur un espace de données. Pour l'utiliser, on doit disposer des cinq éléments suivants :

1) Un principe de codage de l'élément de population. Cette étape associe à chacun des points de l'espace d'état une structure de données. Elle se place généralement après une phase de modélisation mathématique du problème traité. le choix du codage des données conditionne le succès des algorithmes génétiques. Les codages binaires ont été très employés à l'origine. Les codages réels sont désormais largement utilisés, notamment dans les domaines applicatifs, pour l'optimisation de problèmes à variables continues.

2) Un mécanisme de génération de la population initiale. Ce mécanisme doit être capable de produire une population d'individus non homogène qui servira de base pour les générations futures. Le choix de la population initiale est important car il peut rendre plus ou moins rapide la convergence vers l'optimum global. Dans le cas où l'on ne connaît rien du problème à résoudre, il est essentiel que la population initiale soit répartie sur tout le domaine de recherche.

3) Une fonction à optimiser. Celle-ci prend ses valeurs dans R+ et est appelée **fitness** ou fonction d'évaluation de l'individu. Celle-ci est utilisée pour sélectionner et reproduire

les meilleurs individus de la population.

4) Des opérateurs permettant de diversifier la population au cours des générations et d'explorer l'espace d'état. L'opérateur de croisement recompose les gènes d'individus existant dans la population, l'opérateur de mutation a pour but de garantir l'exploration de l'espace d'état.

5) Des paramètres de dimensionnement : taille de la population, nombre total de générations ou critère d'arrêt, probabilités d'application des opérateurs de croisement et de mutation. Le principe général du fonctionnement d'un algorithme génétique est représenté

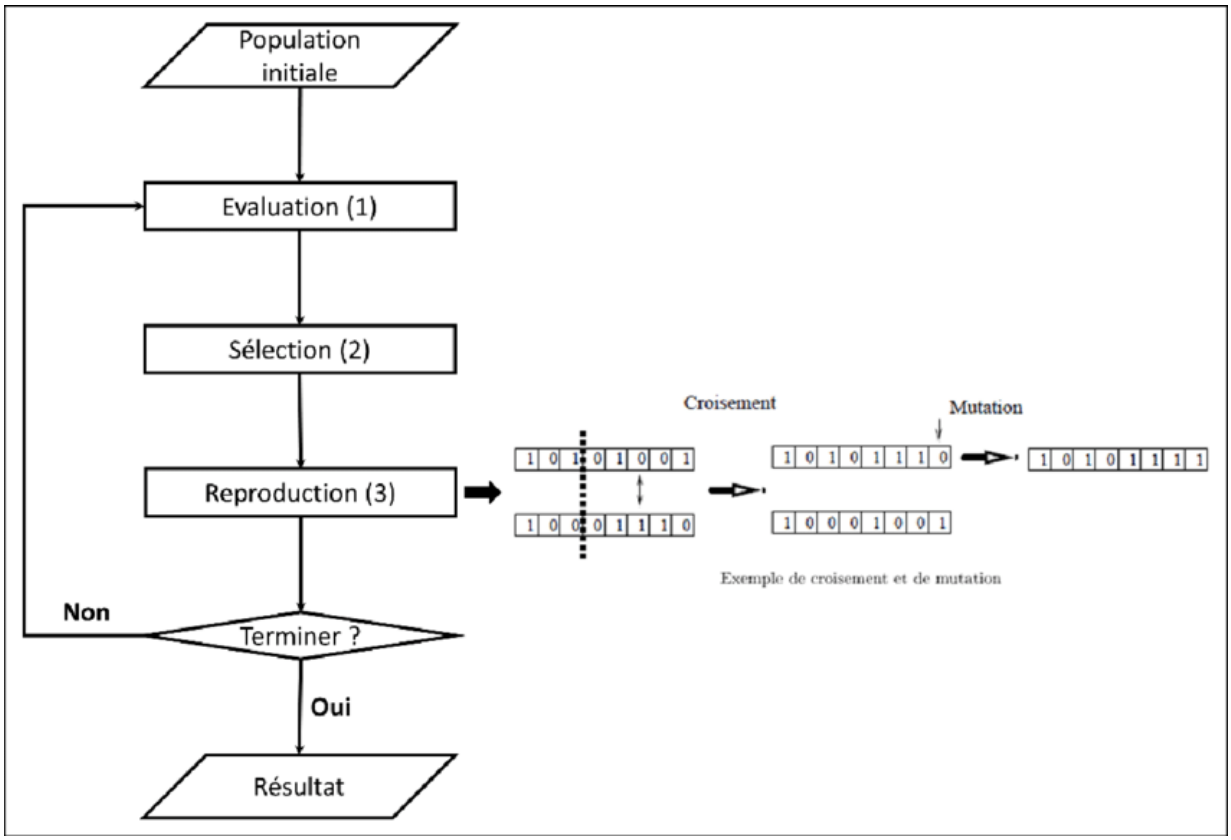


Figure 2.3: Principe de fonctionnement d'un algorithme génétique .

2.5. Conclusion

Dans ce chapitre nous avons présenté un état de l'art sur certain travaux concertant la conception automatique de comportements coordonnés. Nous avons également présenté la théorie des deux algorithmes que nous avons choisi pour réaliser notre travail. Le chapitre suivant sera consacré à la conception de notre application.

Chapitre 3

Conception et implémentation

Chapitre 3 : Conception et implémentation

3.1 Introduction

Ce chapitre se compose de deux phases : la conception et l'implémentation de notre projet.

La phase de conception est le plus important pour pouvoir développer une bonne application, elle a pour objectif de définir les différents composants de l'application qui coopèrent ensemble à la réalisation des fonctionnalités souhaitées. Pour ce la nous aborderons une description générale de notre application ensuite nous mettons en évidence le coté conceptuel de nos applications qui constitue une étape fondamentale qui précède l'implémentation, permet de détailler les différents organigrammes et scénarios à implémenter dans la phase suivante. Ceci permettra de mieux comprendre nos applications.
La phase d'implémentation cette partie a pour objectif de présenter les différents résultats obtenus durant l'étape de réalisation de notre application.
Le but de notre travail consiste à concevoir un comportement coordonné de robots simulés en 2D en utilisant un algorithme de *Flockage* et un algorithme évolutionnaire (AG) pour l'optimisation du comportement obtenu.

3.2 Conception:

3.2.1 Algorithme de Boids

- **Description détaillée**

Chaque boid a un accès direct à toute la description géométrique de la scène, mais le flocage exige qu'il ne réagisse qu'aux copains dans un certain petit quartier autour de lui. Le voisinage est caractérisé par une distance (mesurée à partir du centre du boïd) et un *angle*, mesuré à partir de la direction de vol du boïd. Les copains en dehors de ce quartier local sont ignorés. Le quartier pourrait être considéré comme un modèle de perception limitée (comme par les poissons dans l'eau trouble), mais il est probablement plus correct de le considérer comme définissant la région

dans laquelle les copains influent sur la direction des boids.

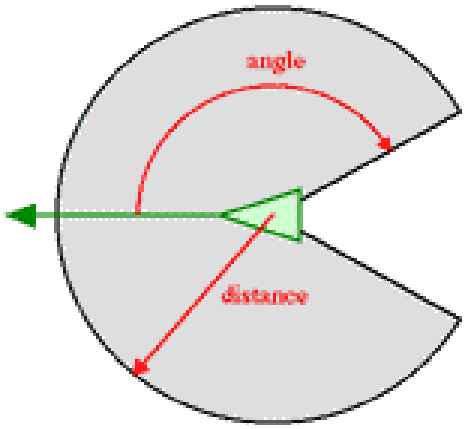


Figure1 : quartier de Boids.

-Le programme Boids se compose d'un groupe de robots (oiseaux) ayant chacun leur position, leur vitesse, et leur orientation.

- Le modèle se compose de trois règles (comportement) de base (séparation, alignement et cohésion)

3.3.2 Séparation:

Chaque Boid essaie d'éviter de se heurter à d'autres Boids. S'il s'approche trop d'un autre Boid, il s'en éloignera, évitant ainsi le surpeuplement.

Il y a plusieurs façons d'appliquer cette règle, une efficace solution pour calculer la séparation ((Sep_i) est par en appliquant l'équation 1.

Les vecteurs définis par la position de boid b_i et de chaque boid visible b_j sont additionné, puis la direction de séparation ($\overrightarrow{Sep_i}$) est calculée comme la somme négative de ces vecteurs.

$$\overrightarrow{Fv_i} = \sum_{\forall b_j \in f} (\overrightarrow{p_i} - \overrightarrow{p_J}) \quad (1)$$

Si seule la règle de séparation est appliquée, alors le troupeau va se dissiper.

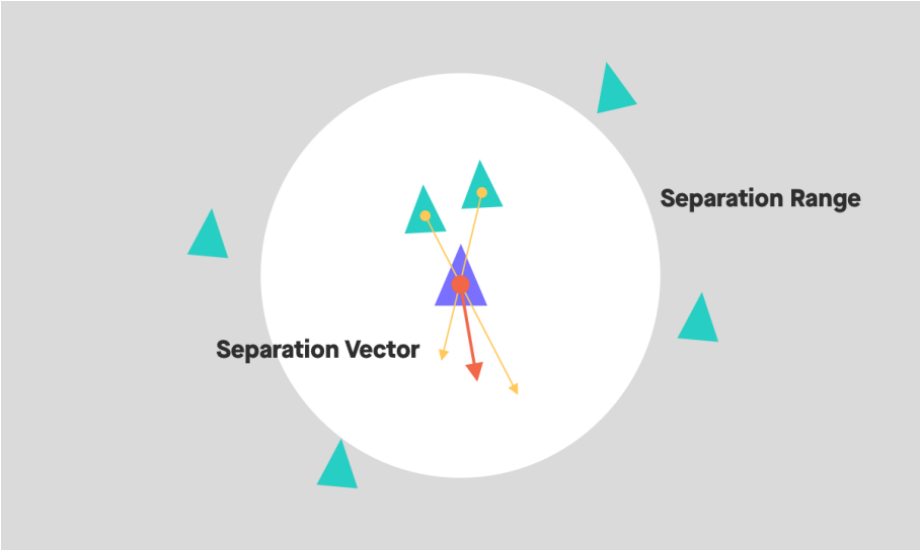


Figure2 : règle de séparation

3.3.3 Alignement :

Les Boids essaient de changer leur position pour qu'elle corresponde à l'alignement moyen des autres oiseaux à proximité. Chaque oiseau vole dans une direction, quand il en voit d'autres, cette règle oblige chaque Boids à essayer d'aligner sa direction en fonction de son voisin immédiat.

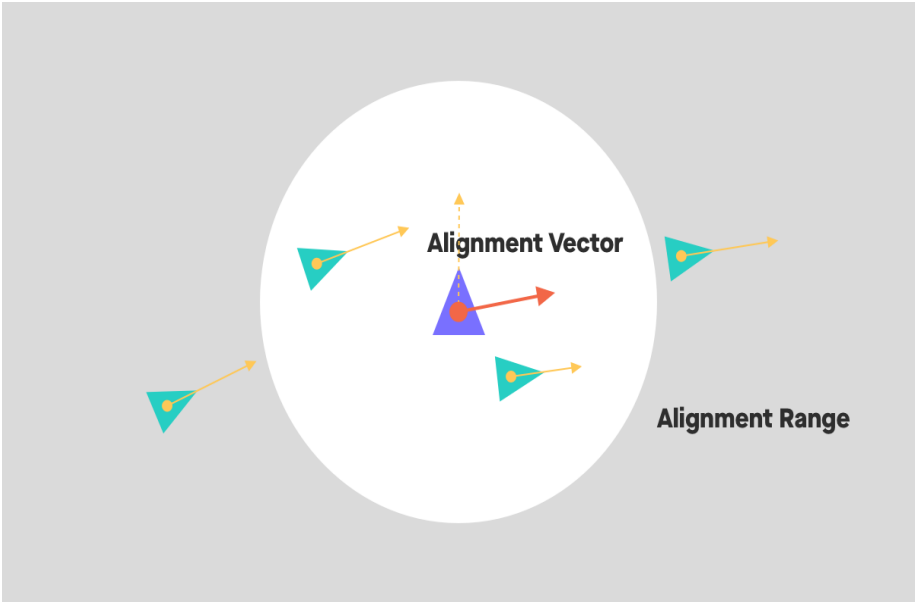
L'alignement (Ali_i)est calculé en deux étapes .Premièrement, le vecteur de vitesse moyenne de compagnons de l'essaim $(\overrightarrow{Fv_i})$ est calculé par l'équation 2.

Alors Ali_i est calculé comme un vecteur de déplacement dans l'équation 3.

$$\overrightarrow{Fv_i} = \sum_{\forall b_j \in f} \frac{\overrightarrow{V_j}}{N} \qquad (2)$$

$$\overrightarrow{Ali_i} = \overrightarrow{Fv_i} - \overrightarrow{V_i} \qquad (3)$$

Ou $\overrightarrow{V_i}$ est le vecteur vitesse de boid i, si cette règle n'était pas utilisée, les boids rebondiraient et ne formeraient pas le beau comportement de flocage que l'on peut voir dans la nature.



Figue3 : règle d'alignement.

3.3.4 Cohésion :

Chaque Boid tente de se déplacer vers la position moyenne des autres boids à proximité, c'est-à-dire que chaque oiseau essaie de rester proche des autres oiseaux de la masse. Lorsqu'ils enregistrent leurs voisins, cette règle essaie d'amener chacun d'eux au centre de l'espace défini par leurs voisins.

Cette règle agit comme le complément de la séparation. Si seule la règle de cohésion est appliquée, tous les boids du troupeau fusionneront en une seule position. La cohésion (coh) du boid (b) est calculée en deux étapes. Tout d'abord, le centre (Fc!) du troupeau (f) qui a ce boid est calculé comme dans l'équation 4.

Ensuite, la tendance du boid à naviguer vers le centre de densité du troupeau est calculée comme le vecteur de déplacement de cohésion comme dans équation 5.

$$\overrightarrow{F_{C_1}} = \sum_{\forall b_j \in f} \frac{\overrightarrow{P_j}}{N} \qquad (4)$$

$$\overrightarrow{coh_i} = \overrightarrow{Fc_i} - \overrightarrow{P_i} \qquad (5)$$

Ou $\overrightarrow{P_J}$ est la position de boid et (N)le nombre total de boids dans (f) .

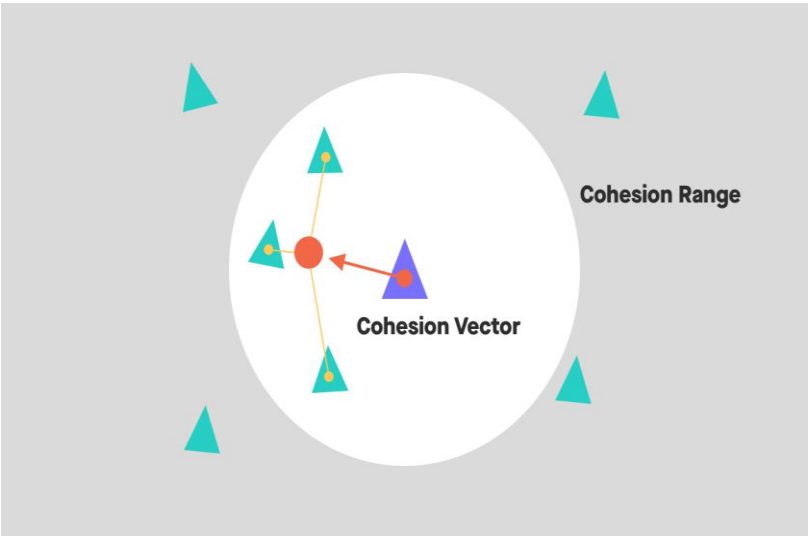


Figure 4 : règle de cohésion

L'algorithme de Boids(Flocking) peut être représenté par le schéma suivant :

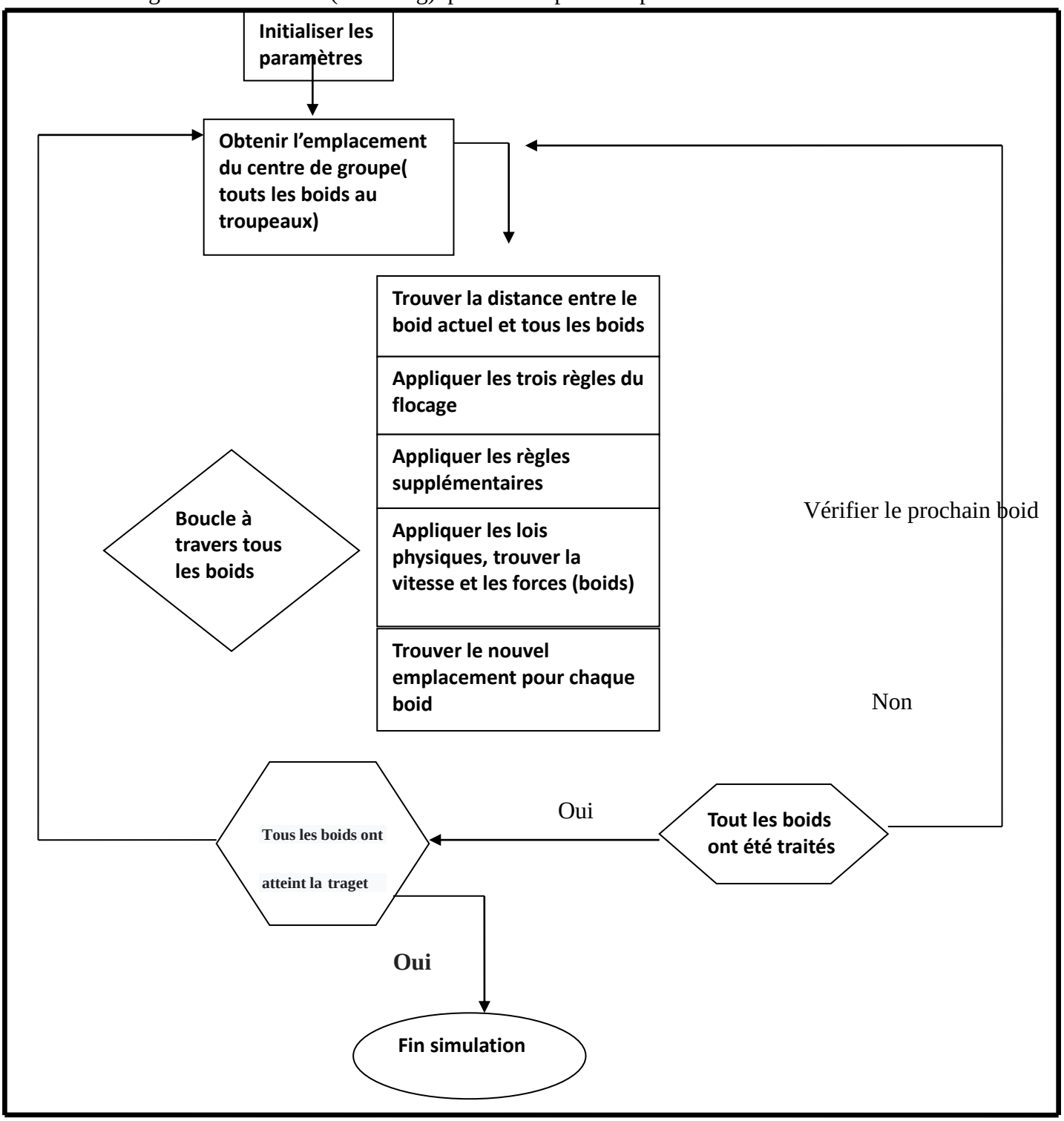


Figure 3.4 : Processus de flocage

3.2.2Algorithmes génétiques

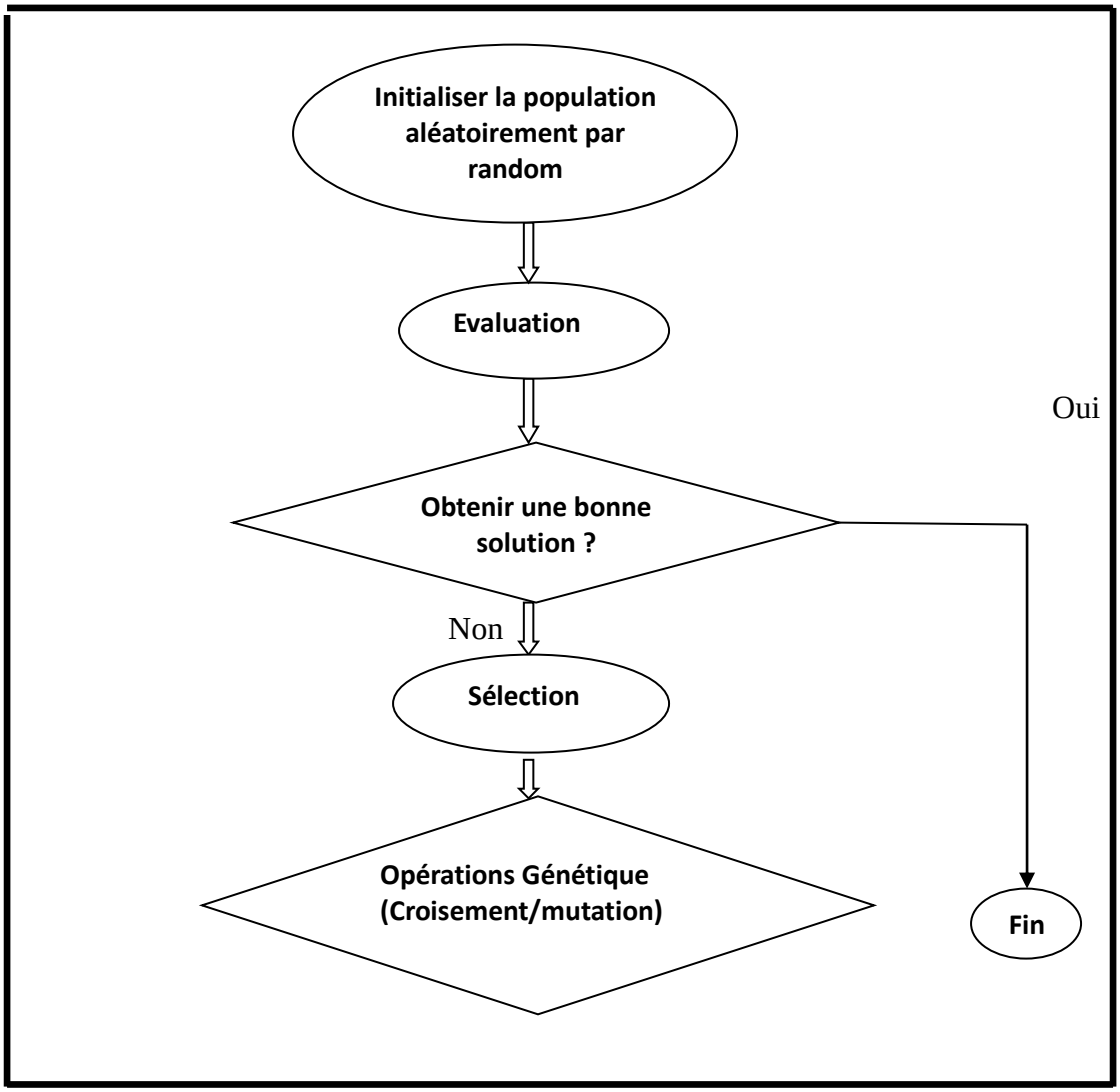


Figure 3.5 : diagramme de AG.

Dans un GA, il est nécessaire de pouvoir évaluer la qualité d'une solution potentielle par rapport à d'autres potentielles solutions. La fonction fitness est responsable de effectuer cette évaluation et renvoyer une valeur de fitness(nombre entier positif) qui reflète le degré optimal de la solution

est. la fonction « fitness » est associée à la fonction objective du problème. La valeur de fitness de l'individu est utilisé pour déterminer la probabilité avec

dont l'individu est sélectionné dans la nouvelle population.
Les méthodes traditionnelles de recherche et d'optimisation sont trop lent à trouver une solution dans une recherche très complexe espace.

GA est une méthode de recherche robuste nécessitant peu d'informations pour rechercher

efficacement dans un grand ou mal espace de recherche compris.

En particulier, une recherche génétique progresse à travers une population de points

contrairement à le point de mire unique de la plupart des algorithmes de recherche.

De plus, il est utile dans le domaine très délicat des problèmes non linéaires.

Les GA ont été utilisés pour résoudre problèmes d'optimisation dans différents domaines tels que

conception robotique, conception technique, automobile, routage optimisé, jeux, etc.

3.3 Implementation

3.3.1 Framework et environnement de travaille

Notre système a été implémenté sous une machine présentant les caractéristiques suivantes :

- Processeur : intel(R) Core(TM) i3-2310M CPU @ 2.10GHz 2.10 GHz
- RAM : 4 GB
-Carte graphique : Intel(R) HD Graphics 4000
- Disque dur : 500 GB

3.3.2 Environnements de développement logiciel :

Afin de mettre en œuvre notre application, l'outil de programmation a été utilisé est : Eclipce IDE for Java Devlopers -2020- 12 comme un IDE (Environnement de développement intégré) et le java comme un langage de programmation.

- **Eclipce** :**Eclipse IDE** est un environnement de développement intégré libre (le terme *Eclipse* désigne également le projet correspondant, lancé par IBM) extensible, universel et polyvalent, permettant potentiellement de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions.

- **Java** : un langage de programmation orienté objet et une plateforme informatique. Créée par l'entreprise Sun Microsystems (souvent juste appelée "Sun") en 1995, et reprise depuis par la société Oracle en 2009, la technologie **Java** est indissociable du domaine de l'informatique et du Web , nous avons utilisés Java version .8.

- Dans notre application nous avons utilisé deux algorithmes principale : l'algorithme de Boids et l'algorithme génétique.
- L'utilisation d'algorithmes génétiques pour améliorer les performances du troupeau Il s'agit d'une **applet Java** qui simule le comportement de Boids. Il est basé sur l'implémentation de traitement de l'algorithme des boids, porté en Java.
- Nous avons implémenté l'algorithme génétique pour faire évoluer le comportement des boids pour que se bouger ou se déplacer d'une manière coordonnée et éviter un obstacle statique.
- Nous avons présenté l'algorithme de boids dans notre programme en deux class : Class boid et class Flock principales et l'algorithme génétique en deux class Class GA et class Population , et une class obstacle.

Algorithme principale de Boids

```
Initialisation de paramètres ;
|
|
|           dessiner_boids()
|
|           Déplacer tous les boids vers de nouvelles positions ()
|
|     Vector v1, v2, v3 ;
|           Boid b ;
|
|           pour (chaque boid b) faire
|           |
|           |       v1 = régle1(b) ;
|           |       v2 = régle2(b) ;
|           |       v3 = régle3(b) ;
|           |
|           |       b.velocity = b.velocity + v1 + v2 + v3 ;
|           |       b.position = b.position + b.velocity ;
|           |
|           Fin pour
|
|       Pm  = (b1.position + b2.position + ... + bN.position) / N ;
|
Fin
```

- **dessiner_boids()** : dessine simplement une 'frame' de l'animation, avec tous les boids dans leurs positions actuelles.

- **Déplacer tous les boids vers de nouvelles positions () :**

Notez qu'il ne s'agit que d'opérations vectorielles simples sur les positions des boïdes. Chacune des règles des boids fonctionne indépendamment, donc, pour chaque boid, nous avons calculé de combien il sera déplacé par chacune des trois règles, nous avons donné trois vecteurs de

vitesse. Ensuite nous avons ajouté ces trois vecteurs à la vitesse actuelle du boid pour déterminer sa nouvelle vitesse. Interprétant la vitesse comme la distance parcourue par le boid par un pas de temps, nous l'ajoutons simplement à la position actuelle.

- **Regle 1** : **Cohésion** (rester proche de ses voisins en se rapprochant du centre de l'essaim)

Supposons que nous ayons N boids, appelés b1, b2, ..., bN. , la position d'un boid b est notée b.position (les positions ici sont des vecteurs).Alors la position moyenne Pm de tous les tous les N boides est donné par :

$$Pm = (b_1.position + b_2.position + ... + b_N.position) / N$$

Cependant, la position moyenne «centre de masse» est une propriété de l'ensemble du l'essaim, ce n'est pas quelque chose qui serait considéré par un boid individuel.

Alors , on a déplacé le boïd vers son « centre perçu », qui est le centre de tous les autres boïds, sans lui-même. Ainsi, pour boidJ (1 <= J <= N), le centre perçu pcJ est donné

$$pc_J = (b_1.position + b_2.position + ... + b_{J-1}.position + b_{J+1}.position + ... + b_N.position) / (N-1)$$

Après avoir calculé le centre perçu, nous avons déterminé comment déplacer le corps vers lui. Pour le déplacer de 1% vers le centre (c'est à peu près le facteur que nous avons utilisé), cela est donné par (pcJ - bJ.position) / 100.
Donc le code de régle1 est :

Procedure regle1(boid b_J)

```
    Vector pcJ
    Pour  (chaque BOID b) faire
        If( b != bJ )
      pcJ = pcJ + b.position ;
    Fin
    pcJ = pcJ / N-1 ;
    RETURN (pcJ - bJ.position) / 100
Fin
```

- **Regle 2 : Séparation**

Le but de cette règle est de permettre aux boids de s'assurer qu'ils ne se heurtent pas les uns aux autres. Je regarde simplement chaque boid, et s'il se trouve à une petite distance définie (disons 100 unités) d'un autre boid, déplacez-le à nouveau aussi loin qu'il l'est déjà. Cela se fait en soustrayant d'un vecteur c le déplacement de chaque boide qui se trouve à proximité.
Nous initialisons c à zéro car nous voulons que cette règle nous donne un vecteur qui, lorsqu'il est ajouté à la position actuelle, éloigne un boid de ceux qui l'entourent

Procedure regle2(boid b_J)

```
  Vector c = 0;
  Pour ( chaque Boid b)
    If ( b != bJ )
      if  (|b.position - bJ.position| < 100)
       c = c - (b.position - bJ.position) ;
    fin if
  Fin pour
```

Return c ;

Fin

- **Regle 3 : Alignement**

Les Boids essaient de faire correspondre la vitesse avec les Boids proches. Cette régle est similaire à la règle 1, mais au lieu de faire la moyenne des positions des autres boids, nous faisons la moyenne des vitesses. Nous avons calculé une « vitesse perçue », pvJ, puis ajoutons une petite partie à la vitesse actuelle du boid.

Procédure regle3 (boid b_J)

Vector pv_J ;

Pour (chaque Boid b) faire

if (b != b_J)

$pv_J = pv_J + b.velocity$;

fin if

fin pour

$pv_J = pv_J / N-1$:

Return (pv_J - b_J.velocity) / 8

Fin

❖ **Les déférents paramètres utilisés dans notre programme :**

Paramètres	**Description**
Position	Actuel position de boid
velocity	Actuel vitesse de boid
acceleration	Accélération acquise par un boid dans chaque unité de temps
MACFORCE	= 2.5 force maximale pouvant agir sur un

	boid
MAXSPEED	= 0.1 Vitesse maximale, vitesse qu'un boid peut atteindre

Tableau 3.1: Paramètres utilisé dans la clas Boid.

Paramètres	**Description**
ArrayList<Boid>boids;	Tableau de boids
xt,yt [];	Tableaux de positions boids
lifespan	= 1000 Temps entre chaque itération
pop SIZE = 1000	Nombres de boids dans le système
WIDTH = 150	Largeur de la fenêtre
HEIGHT = 700	longueur de la fenêtre

Tableau 3.2: les paramètres de class Flock.

paramètres	**valeurs**
Taille de population	100
Nombre de génération	initialiser a 0
Individu	Chaque individu représenté par les trois régle de flocage en décimal
Taux de mutation	0.05
Taux de croisement	0.5

Tableau 3.3: Paramètres de AG.

❖ **Description détaillée de méthode de fonction fitness :**

```
private void fitness(ArrayList<Boid> guppies, Obstacle p, int gener) {
  float safetyDist = 50f;   ⇦   la distance de sécurité
 fitness += safetyDist;   ⇦   incrémenter la fitness par la distance de l'obstacle
   for (Boid other : guppies) {   ⇦   vérifier la distance entre autres boids
```

```
float touching = PVector.dist(this.position, other.position);

    if (touching <= 10) {
    this.fitness /= 2; }        } punir les boids qui se heurtent

    if (this.velocity.mag() <= 1){
     this.fitness /= 2; }       } punir les boids qui ne bougent pas

float sep = Math.abs(2.3f - this.getGene("separate"));
float ali = Math.abs(1.2f - this.getGene("align"));
float coh = Math.abs(1.4f - this.getGene("cohesion"));

float score = sep + ali + coh;

 fitness = fitness / gener;          ⇐ les boids qui ont la bonne note
                }
        }

public void randomGenes()   ⇐ définit des gènes aléatoires

{
                HashMap<String, Float> geneMap = new HashMap<String, Float>();
                geneMap.put("align", (float) Math.random()*2);
                geneMap.put("separate", (float) Math.random()*2);
                geneMap.put("cohesion", (float) Math.random()*2);

                this.setGenes(geneMap);

}

 public void setGenes(HashMap<String, Float> geneMap) {
                this.genes = geneMap;

}
public HashMap<String, Float> getGenes() {
                return this.genes;
        }
```

- **HashMap :** une liste , comme un dictionnaire, qui contient la représentation de chaque individu (les trois règles de flocage : alignement, séparation, cohésion et leurs valeurs).

❖ **Méthode sélection** :

```
 Initialisation : P1 , P2 .
début
 pour( i=1  à la taille de population) faire
     select P1 random ;
     select P2 random ;
  croisement (P1,P2) ;⇐      nouveau boid
       nv.mutation ;
     Fils_liste_add(nv) ;
  Fin pour

Fin
```

❖ **Méthodes croisement**

```
Initialisation: Taux croisement = 0.05
Début
       Choisir valeur aléatoire ;
    If (valeur random > Taux croisement)
         Boid b1 .mating  (Boid b2) ;
      e lse
      Boid b2 .mating (Boid b1) ;

     Fils_liste_add(nv) ;
   Fin

Fin
```

3.4 Résultats :

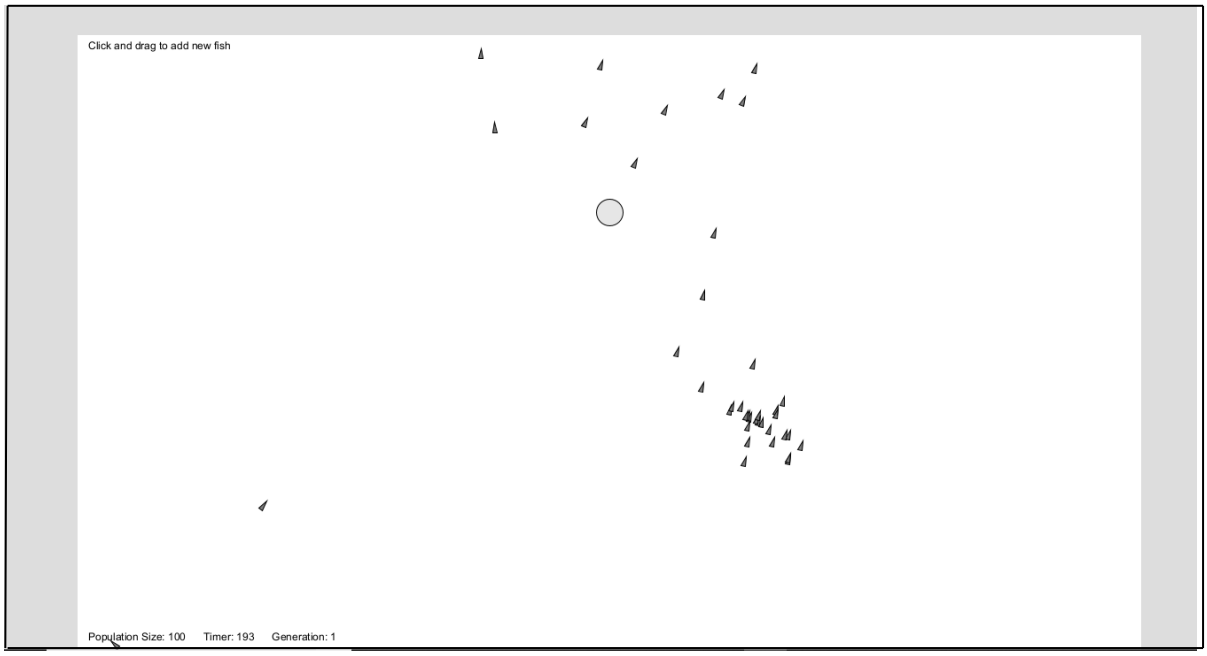


Figure 7 : simulation de notre application (génération 1, time 193).

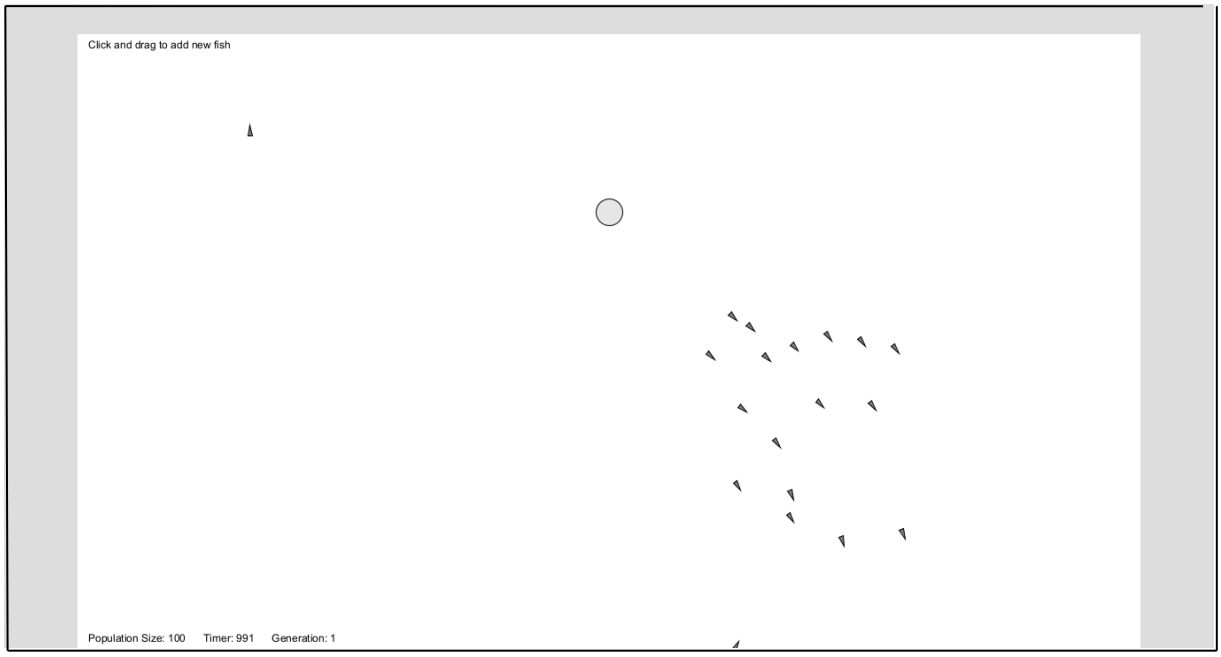


Figure 8 : simulation de notre application (génération1, Time 991).

- Ici dans le cas de génération 1(figure 7,8), nous avons observé que les robots (boids) ont évités l'obstacle, mais il y a un boid loin de ces voisins (l'essaim), pas de coordination, donc elle n'est pas une bonne génération.

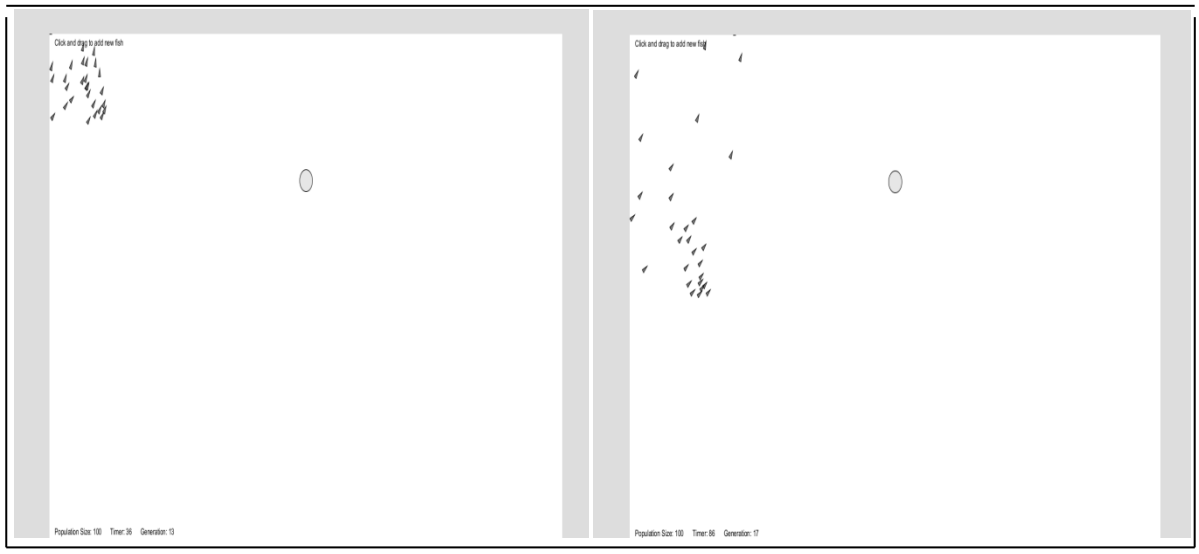


Figure 9: simulation de notre application (tous les robots en essaim).

- Dans les générations 13 et 17, nous avons remarqué que tous les boids sont dans l'essaim, et c'est ce que nous souhaiterons à arriver dans ce travail.

3.5 Conclusion :

Nous avons présenté dans ce chapitre la mise en œuvre de notre application qui permet de faire un comportement coordonnée, on a utilisé l'algorithme de Boids qui est basé sur les trois règles de base (séparation, alignement et cohésion) pour avoir un comportement coordonné. D'un autre coté nous avons utilisé l'algorithme génétique pour l'optimisation du comportement obtenu pour éviter les obstacles. Les résultats obtenus avec notre travail sont des simulations de comportement coordonné de robots en 2D qui avec l'optimisation de l'AG les robots ont pu se déplacer en groupe et éviter au même temps les obstacles.

Conclusion générale

La robotique en essaim est une domaine de recherche très intéressant et ambitieux qui cherche principalement à intégrer la théorie derrière les essaims dans la nature à des systèmes robotiques.

Les modèles de robotique en essaim sont fondamentalement inspirés des comportements collectifs d'animaux sociaux tels que les oiseaux, les fourmis et les abeilles. Ils sont généralement appliqués dans les études traitant de problèmes de tâches collectives. À titre d'exemple, les schémas de formation et les schémas d'agrégation spécifiquement auto-organisation font partie des problématiques complexes qui intéressent la littérature sur la robotique en essaim.

Dans ce mémoire, nous avons utilisé et implémenté l'algorithme de Boids (flocking) pour la conception d'un mouvement coordonné de robots en essaims. Nous avons optimisé ce comportement par l'utilisation des algorithmes génétiques pour que les robots se déplacent touts ensemble tout en évitant un obstacle statique dans un environnement de simulation bi-dimensionnel.

Bibliographie

[1] Beekman M., Sword G.A., Simpson S.J., "BiologicalFoundations of Swarm Intelligence", chapitre Swarm Intelligence. Part of the series Natural ComputingSerie, pp 341, 2008

[2] Abraham A., Guo H. and Liu H., " Swarm Intelligence: Foundations, Perspectives and Applications ", chapitre Swarm Intelligence Vol.26 of the serieStudies in computational Intelligence pp. 3-25, 2006

[3] El-Sayed H., Belal M., Almojel A. and Gaber J., "Swarm Intelligence", Handbook of Bioinsiredalgorithms and application. 2005

[4] Martens D., Baesens B. and Fawcett T., " Editorial survey: swarm Intelligence for data mining ", Machine Learning, Vol. 82, Issue 1, pp 1-42, 2011.

[5] OnurSoysal et ErolSahin. "Probabilisticaggregationstrategies in swarmroboticsystems". Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE. IEEE. 2005, pp. 325–332.

[6] Marco Dorigo, Mauro Birattari, et al. "Swarm intelligence". Scholarpedia 2.9(2007), p. 146. NikolausCorrell et AlcherioMartinoli. "Modeling and designing self-organizedaggregation in a swarm of miniature robots". The International Journal of RoboticsResearch 30.5 (2011), pp. 615–626.

[7] ManueleBrambillaetal."Swarmrobotics:areviewfromtheswarmengineering perspective". In: Swarm Intelligence 7.1 (2013), pp. 1–41.

[8] Reynolds, C. W. (1987). Flocks, herds, and schools : A distributed behavioral model. Computer Graphics, 21(4):25–34.

[9] Reeves, W. T. (1983). Particle systems-a technique for modeling a class of fuzzy objects. In acm Transactions on Graphics, volume 2 de acm SIGGRAPH ‘83 Proceedings, pages 359–376.

[10] Reynolds, C. W. (1988). Not bumping into things.

[11] B. P. Gerkey and M. J. Matarić, “Sold!: Auction methods for multirobot coordination,” IEEE Trans. Robot. Autom., vol. 18, no. 5, pp. 758–768, 2002.

[12] G. A. Kaminka, R. Schechter-glick, and V. Sadov, “Using Sensor Morphology for Multirobot Formations,” vol. 24, no. 2, pp. 271–282, 2008.

[13] V. Garg and M. Jhamb, “A Review of Wireless Sensor Network on Localization Techniques,” Int. J. Eng. Trends Technol., vol. 4, no.April, pp. 1049–1053, 2013.

[14] A. Wichmann, B. D. Okkalioglu, and T. Korkmaz, “The integration of mobile (tele) robotics and wireless sensor networks: A survey,” Comput.Commun., vol. 51, no. September, pp. 21–35, 2014.

[15] P. Corke, R. Peterson, and D. Rus, “Localization and navigation assisted by networked cooperating sensors and robots,” Int. J. Rob. Res., vol. 24, no. 9, pp. 771–786, 2005.

[16] M. Schwager, J. McLurkin, and D. Rus, “Distributed Coverage Control with Sensory Feedback for Networked Robots.,”Robot. Sci. Syst., no.June 2014, pp. 49–56, 2006.

Annexe de l'application

Dans notre application on a utilisé l'algorithme de Boids qui est basé sur les trois règles de base (séparation, alignement et cohésion) pour avoir un comportement coordonné. On a exprimé cet algrorithme dans deux classes « **Class boid** et **Class Flock** ».

D'un autre coté nous avons utilisé l'algorithme génétique pour l'optimisation du comportement obtenu pour éviter les obstacles. « **Class GA , Class Population , Class Obstacle** ».

Les résultats obtenus avec notre travail sont des simulations de comportement coordonné de robots en 2D qui avec l'optimisation de l'AG les robots ont pu se déplacer en groupe et éviter au même temps les obstacles.

```
public class Boid extends GA {

      // variable declarations
      PVector position;
      PVector velocity;

      float maxforce;
      float maxspeed;
      PApplet parent;


      float fitness;

      //boolean eaten;


    int[] xt,yt;
    static final int N = 3;

    //Coordinates to draw a triangle
    private static final int[] X = new int[] {0, 3, 6};
    private static final int[] Y = new int[] {0, -12, 0};



      // Auto-generated constructor stub
      public Boid(PApplet p, int x, int y, GA dna, float fit) {

            // PApplet to reference canvas
```

```
      parent = p;
      if ((x==0) && (y==0)) {

            // random start position if no input variables
            position = new PVector(0,0);
      } else {
            position = new PVector(x,y);
      }
      // random velocity vector
      velocity = new PVector(p.random(-1,1),p.random(-1,1));
      //*r = -10.0f;
      // maximums
      maxspeed = 2.5f;
      maxforce = 0.1f;

      // if incoming dna is null, assign random genes
      if (dna != null) {
            this.setGenes(dna.genes);
      } else {
            this.randomGenes();
      }


   xt = new int[N];
   yt = new int[N];
}

// applies various swarm forces

protected void school(ArrayList<Boid>Boids, Obstacle p) {
      // init. each force
      PVector a = align(Boids);
      PVector s = separate(Boids, p);
      PVector c = cohesion(Boids);

      // apply weights
      a.mult(this.getGene("align"));
      s.mult(this.getGene("separate"));
      c.mult(this.getGene("cohesion"));
      //*f.mult(this.getGene("flight"));
      // apply each force
      applyForce(c);
      applyForce(s);
      applyForce(a);
      //*applyForce(f);
}



// run method for Boids
public void run(ArrayList<Boid> Boids, Obstacle p, int gen) {
      school(Boids, p);
```

```
            update();
            fitness(Boids, p, gen);
            borders();
            render();
      }



      // continuously calculates a Boid's fitness score
      private void fitness(ArrayList<Boid>Boids, Obstacle p, int gener)
{

            float safetyDist = 50f;


            //boolean closeCall = (safetyDist <= 30);



                  // increment fitness by distance from pred
                  fitness += safetyDist;

                  // check distance from other guppies
                  for (Boid other :Boids) {
                        float touching = PVector.dist(this.position,
other.position);
                        // punish Boids that collide
                        if (touching <= 10) {
                              this.fitness /= 2;
                        }
                  }

                  // punish Boids that don't move
                  if (this.velocity.mag() <= 1) {
                        this.fitness /= 2;
                  }

                  // at end of generation, score based on preset values
                  if (gener == 15) {
                        float sep = Math.abs(2.3f -
this.getGene("separate"));
                        float ali = Math.abs(1.2f -
this.getGene("align"));
                        float coh = Math.abs(1.4f -
this.getGene("cohesion"));

                        float score = sep + ali + coh;

                        // reward high-scoring Boids
                        if (score < 1) {
                              this.fitness *= 10^8;
                        } else if (score < 5) {
```

```
                    this.fitness *= 10^6;
               } else if (score < 10) {
                    this.fitness *= 10^4;
               } else if (score < 20) {
                    this.fitness *= 10^2;
               } else if (score < 30) {
                    this.fitness *= 10;
               } else if (score < 50) {
                    this.fitness *= 5;
               } else if (score < 75) {
                    this.fitness *= 3;
               } else if (score < 100) {
                    this.fitness *= 2;
               }


               fitness = fitness / gener;
     }
}


// keeps Boids moving in similar direction

private PVector align(ArrayList<Boid>Boids) {

  // PVector to hold sum of direction

     PVector sum = new PVector(0,0);
     // counter to divide by
     int count = 0;
     // get pull gene for comparison
//*  float pullDist = this.getGene("pull");
     // loop through all fish
     for (Boid other :boids) {
          // measure distance between current boids and others
          float d = PVector.sub(position,other.position).mag();
          // if between 0 and pull distance
          if ((d > 0) && (d < 50f)) {
               // add vector of neighbor to sum
               sum.add(other.velocity);
               count++;
          }
     }
     // if other boid w/in pull distance
     if (count > 0) {
          // divide sum by count
          sum.div((float)count);
          sum.normalize();
          sum.mult(maxspeed);
          // get new steering vector
          PVector steer = PVector.sub(sum,velocity);
          steer.limit(maxforce);
```

```
                return steer;
           } else {
                return new PVector(0,0);
           }
     }

     // steers away from fish that are too close
     private PVector separate(ArrayList<Boid> Boids, Obstacle p) {
           PVector steer = new PVector();
           int count = 0;

           // loop through all Boids
           for (Boid other :Boids) {
                // check distance like in align()
                float d = PVector.sub(position, other.position).mag();
                // if d is within range
                if ((d > 0) && (d < 35f)) {
                     // get vector away from neighbor
                     PVector diff = PVector.sub(position,
other.position);
                     diff.normalize();
                     // weight by distance
                     diff.div(d*2);
                     steer.add(diff);
                     count++;
                }
           }


        double d = PVector.sub(this.position,p.position).mag();

        if(d>0 && d<35f){
            // Calculate vector pointing away from neighbor
            PVector diff = PVector.sub(this.position,p.position);
            diff.normalize();
            diff.div((float)d);
            diff.mult(2);       //Multiply by a factor to give effect
of a predator
            steer.add(diff);
            count++;
        }


           // get average
           if (count > 0) {
                steer.div(count);
           }
           // if there is a direction
           if (steer.mag() > 0) {
                // steering = desired - velocity
                // Reynolds
                steer.normalize();
```

```
            steer.mult(maxspeed);
            steer.sub(velocity);
            steer.limit(maxforce);
        }
        return steer;
    }

    // steers fish towards center of mass
    private PVector cohesion(ArrayList<Boid> guppies) {
        PVector sum = new PVector(0,0);
        int count = 0;
    //*  float pullDist = this.getGene("pull");
        // loop thru all fish
        for (Boid other : guppies) {
            // if within pull distance (
            float d = PVector.sub(position, other.position).mag();
            if ((d > 0 ) && (d < 50f)) {
                sum.add(other.position);
                count++;
            }
        }
        if (count > 0 ) {
            sum.div(count);
            return seek(sum);
        } else {
            return new PVector(0,0);
        }
    }




    // draws the Boid
    protected void render() {



        //float theta = (float) (velocity.heading() + (Math.PI/2)) ;
        parent.pushMatrix();
        //parent.stroke(0);
        parent.fill(127);
        //parent.translate(position.x, position.y);
        //parent.rotate(theta);


      double theta = Math.atan2(velocity.y,
velocity.x+Math.toRadians(90));
      double cos = Math.cos(theta);
      double sin = Math.sin(theta);

      for(int i=0;i<N;i++){
```

```
            xt[i] = (int)(X[i]*cos - Y[i]*sin) + (int)position.x;
            yt[i] = (int)(X[i]*sin + Y[i]*cos) + (int)position.y;
        }


          parent.triangle(position.x+xt[0], position.y+yt[0],
position.x+xt[1],position.y+yt[1],
position.x+xt[2],position.y+yt[2]);
          parent.popMatrix();
     }


     // applies a PVector acceleration
     protected void applyForce(PVector force) {
          velocity.add(force);
     }

     // method to change direction
     protected void update() {
          position.add(velocity);
     }

     // border behavior

     protected void borders() {

      // wraps boid around borders

        double x = position.x;
        double y = position.y;
        if (x < 0) x = 1200;
        if (y < 0) y = 700;
        if (x > 1200) x = 0;
        if (y > 700) y = 0;
        position.x = (float) x;
        position.y = (float) y;
     }


     // applies a steering force towards a target
     // STEER = DESIRED MINUS VELOCITY
     public PVector seek(PVector target) {

          target.sub(position);
          target.normalize();
          target.mult(maxspeed);
        PVector steer = target.sub(velocity);
        steer.limit(maxforce);  // Limit to maximum steering force

        return steer;
```

```
	}


}

*********************************************************************************


public class Flock extends PApplet {
	// initialize objects
	Population pop;
	Obstacle obs;
	GA dna;
	// variable set
	int popSize = 100;  //nbr des boids
	int lifespan = 1000;
	int generation = 0;
	int timer = 0;


	// PApplet extension
	public static void main(String[] args) {
		//PApplet.main("Lir"); flock
		PApplet.main(new String[] { "--present",
Flock.class.getName() });
	}

	// canvas size
	public void settings() {
		size(1200,700);
	}


	public void setup() {
		pop = new Population(this);
		for (int i = 0; i < popSize; i++) {
			Boid g = new Boid(this, 0, 0, dna, 0);
			pop.addboid(g);
		}
		// add obs
		//
		obs = new Obstacle(this, 600, 200);
	}

	// draw canvas and run
	public void draw() {
		background(255);
		int displayPopSize = pop.run(obs, lifespan);
		// add a predator
		obs.run(pop);
		// increment timer
```

```
            timer++;
            // process generation
            if (timer == lifespan) {
                  pop.eval();
                  pop.naturalSelection(popSize);
                  timer = 0;
                  generation++;
            }
            // help text
            fill(0);
            text(("Population Size: " + displayPopSize + "      Timer: "
+ timer + "      Generation: "
                        + generation), 12, this.height - 16);

      }


}


**********************************************************************


public class GA {

      float mutationRate = 0.05f;

      // initialize empty map of genes
      HashMap<String, Float> genes = new HashMap<String, Float>();

      // sets random genes within range
      public void randomGenes() {

      HashMap<String, Float> geneMap = new HashMap<String, Float>();
            geneMap.put("align", (float) Math.random()*2);
            geneMap.put("separate", (float) Math.random()*2);
            geneMap.put("cohesion", (float) Math.random()*2);

            this.setGenes(geneMap);
      }
      // setter for full gene map
      public void setGenes(HashMap<String, Float> geneMap) {
            this.genes = geneMap;
      }

      // getter for full gene map
      public HashMap<String, Float> getGenes() {
            return this.genes;
      }

      // getter for single gene value
```

```
	public Float getGene(String key) {
		return genes.get(key);
	}

	// function for mating two Boids
	public GA mating(Boid mate) {
		// init empty DNA for child
		GA childDNA = new GA();
		// flip a coin
		double coin = Math.random();
		// create new map for child genes
		HashMap<String, Float> childGenes = new HashMap<String,
Float>();
		// iterate thru parent genes
		for (HashMap.Entry<String, Float> entry :
mate.genes.entrySet()) {
			// get gene values
			String key = entry.getKey();
			Float val = entry.getValue();
			// 50% chance of gene from parent 1
			if (coin >= 0.5) {
				childGenes.put(key, val);
			} else {
				// 50% chance from parent 2
				childGenes.put(key, mate.getGene(key));
			}
			// re-randomize
			coin = Math.random();
		}
		// add genes to child DNA
		childDNA.setGenes(childGenes);
		return childDNA;
	}

	public void mutation() {
		// iterate thru genes
		for (HashMap.Entry<String, Float> entry :
this.genes.entrySet()) {
			// for a small percentage of genes
			if (Math.random() < mutationRate) {
				String key = entry.getKey();
					this.genes.put(key, (float)
Math.random()*2);
			}
		}
	}
}
```

****

```
public class Population {

      // initialize variables
      ArrayList<Boid> boids;       // la population
      ArrayList<Boid> matingPool;
      ArrayList<Boid> fils;
      PApplet parent;
      float minFitness;

     //String filepath;
      //String line;


      // construct array boids
      public Population(PApplet p) {
            boids = new ArrayList<Boid>();
            parent = p;
      }

      // method to add Boids to array
      public void addboid(Boid g) {
            boids.add(g);
      }

      int run(Obstacle p1, int gen) {
            int popSize = 0;
            for (Boid g : boids) {
                  g.run(boids, p1, gen);
                  popSize++;
            }
            return popSize;
      }

      // evaluate a population of Boids
      float eval() {
            // init mating pool
            matingPool = new ArrayList<Boid>();
            // init max fitness for normalization
            float minFitness = 0;
            // loop thru boids and get fitness score
            for (Boid g : boids) {
                  // make sure max is highest
                  if (g.fitness > minFitness) {
                        minFitness = g.fitness;
                  }
            }
            // loop thru Boids and normalize by max
            for (Boid g : boids) {
                  if (minFitness != 0) {
                        g.fitness /= minFitness;
                  }
```

```
            }
            // loop thru    boids and add to mating pool
            for (Boid g : boids) {
                  if (g.fitness > 0) {
                        // higher fitness scores get more entries in
mating pool
                        int n = Math.round(g.fitness * 100);
                        for (int i=1; i<n; i++) {
                              matingPool.add(g);
                        }
                  }
            }
            return minFitness;
      }



      // create a new generation of Boids
      void naturalSelection(int popsize) {
            // init offspring list
            fils = new ArrayList<Boid>();
            // iterate thru population
            int n = popsize;
            // use standard for loop to avoid concurrent modification
            for (int i=0; i < n; i++) {
                  // get a random boid from mating pool
                  int randomMate =
(int)(Math.random()*matingPool.size());
                  Boid parent1 = this.matingPool.get(randomMate);
                  // find a mate
                  randomMate = (int)(Math.random()*matingPool.size());
                  Boid parent2 = this.matingPool.get(randomMate);
                  // create a new child guppy
                  GA childDNA = parent1.mating(parent2);
                  Boid child = new Boid(parent, 0, 0, childDNA, 0);
                  child.mutation();
                  // add to offspring pool
                  fils.add(child);
            }
            // replace current population with new one
            this.boids.clear();
            this.boids.addAll(fils);
      }

}
```

```
public class Obstacle{


	// variable declarations
	PVector position;
	PApplet parent;


	// Auto-generated constructor stub
	public Obstacle(PApplet p, int x, int y) {

		// PApplet to reference canvas
		parent = p;
		position = new PVector(x,y);
	}

	public void render() {

		parent.pushMatrix();
		parent.fill(230);
		parent.ellipse(position.x,position.y,30,30);
		parent.popMatrix();
	}

	public void run(Population pop) {
		render();
	}


}
```