# Projet Robocode

Encadré par : **Ludovic Bonnefoy**

Membres du projet :**Gallian Colombeau**, **Laurent Crépin**, **Sofiane Stamboul**, **Hugo Rovelli**

UNIVERSITÉ D'AVIGNON
UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

# Projet Robocode

# I. Introduction

Robocode est un jeu vidéo qui a été distribué par IBM, à des fins éducatives. En effet, son but principale étant l'apprentissage du langage de programmation JAVA.

Le fonctionnement du jeu est simple, des robots virtuels s'affrontent sur un champ de bataille. Chaque robot se déplace, détecte et tire sur l'adversaire. La partie peut se dérouler en plusieurs rounds sachant que chacun d'eux se termine dès qu'il ne reste qu'un seul robot.

Aujourd'hui, IBM ne maintient plus Robocode, mais il existe une communauté active qui maintient le projet et propose de nouveaux robots. Chaque programmeur peut ainsi tester son robot possédant sa propre Intelligence Artificielle (IA). Il existe un système de tournois qui définie un classement en fonction de la catégorie dans laquelle le robot concourt.

Dans le cadre du projet Master première année, l'objectif principal est d'implémenter une IA qui permet de faire fonctionner notre propre robot. De plus cela nous permettra d'améliorer nos compétences dans le langage que l'on a choisi, ainsi que de découvrir et si possible de maîtriser les algorithmes d'intelligence artificielle.

Dans cette optique, nous nous sommes divisés en deux groupes afin d'aborder deux approches différentes de conception d'une IA pour Robocode. A ce moment là du projet, on peut envisager d'utiliser les deux approches de façon complémentaire ou de concevoir deux robots différents pouvant être concurrent.

Après avoir avoir développé la problématique, nous allons voir les outils Robocode officiels mis à disposition avant d'aborder en détail les deux approches choisies.

# II. La problématique

Rappelons que le but principal du projet est de développer une IA pour un robot virtuel utilisant les sources de Robocode. On est parti sur deux hypothèses différentes pour chaque sous groupes.

## *1. Les hypothèses*

Sachant que le robot se déplace, détecte et tire, le but est de produire une IA qui va utiliser une partie ou la globalité des fonctionnalité du robot. Le but principale de l'IA étant de survivre pour gagner, elle détermine ainsi la stratégie du robot qui peut être agressif, ou défensif et l'idéal étant la combinaison des deux comportements, par rapport à l'environnement.

## *2. Les approches*

De ces hypothèses, nous avons opté pour deux approches différentes. Le premier groupe, composé de Colombeau Gallian et Crepin Laurent s'est orienté sur le coté agressif du robot, en développant une IA optimisant principalement les fonctionnalités de tir du robot. Après moult réflexions, nous avons choisi d'utiliser l'Algorithme Génétique comme méthode d'optimisation. En effet, nous étions déjà intéressé par l'étude de cet algorithme et ce projet nous a donc donné une excellente opportunité de pouvoir l'utiliser.

L'autre groupe, constitué de Stamboul Sofiane et Rovelli Hugo, a adopté une approche différente, se basant sur l'aspect défensif du robot avec une méthode d'apprentissage. Ce dernier s'orientera sur les fonctionnalités de mouvement du robot. Le Réseau de Neurones est l'algorithme d'apprentissage adopté pour traiter cette partie. Étant intéresse et curieux de connaître le fonctionnement d'un algorithme d'apprentissage, le choix s'est fait de lui même. Mais également le fait que les robots les plus performants sont développés grâce a celui-ci.

# III. Robocode IHM et API

## *1. Interface utilisateur*

L'API java de Robocode ce compose de trois modules:

•**Robot API:** contenu dans le package robocode :

Robot API est utilisé pour développer des robots,c'est la seule partie de l'API ou les robots ont les droit d'accès .

La classe la plus basique de cette API est donc la classe Robot qui permet de crée son propre robot les méthode de cette classe permettront de le contrôler

Plusieurs classes dérive de la classe Robot dont AdvancedRobot qui *vous* donne le contrôle total , sur la liste des événements, vous pouvez contrôler l'ordre, voir quand vous ratez un tir, etc...

La classe TeamRobot qui hérite également de robot représente le robot d'équipe cette classe vous permettra de crée plusieurs robot en désignant un chef qui aura la capacité d'envoyer des message a ces allies pour attaquer un ennemie détecter

•**Robot Interface:** contenu dans le package Java robocode.robotinterfaces .

Les interfaces robot sont utilisés pour développer de nouveaux types de robots avec une API différente que l'API Robot standard.

Remarque: Les règles du jeu et les comportements du robot ne peuvent pas être changés.

•**Robot Contrôle:** contenu Dans le package robocode.control

L'API de contrôle est utilisé pour permettre à une autre application de démarrer des batailles de robots et de récupérer les résultats correspondant. Il est également possible d'obtenir des clichés de robots et de balles (comme la position, le cap, niveau d'énergie, etc) à un moment précis dans une bataille.

Cette api a plusieurs spécificité nous on citerons quelque une ci-dessous:

1. toutes les coordonnes sont exprimées en (x,y)
2. l'unité de mesure des distance est le pixel
3. Les coordonnes sont positives
4. point de départ des coordonnes (0,0) est en bas a gauche de l'écran

## Ordre de fonctionnement de Robocode :

- Champ de bataille rafraîchis (re-paint)
- Tous les robots exécuter leur code puis ce mettent en pause.
- Le temps est mis a jour
- Toutes les balles se déplace en vérifiant les collisions. Cela inclut balles tiré pas les ennemis .
- Tout les robots se déplacement .
- Tout les robots font des scannes
- tout les robots préparent de nouvelle action.
- Chaque robot traite sa file d'attente d'événement

# IV. Réseau de neurones

## *1. Présentation*

Au cours de la dernière décennie, les algorithmes d'apprentissage statistique ont suscité beaucoup d'intérêt dans le milieu académique et au sein d'entreprises de diverses industries. Ils ont été implantés avec succès pour l'accomplissement de tâches prédictives reliées à des processus statistiques observés pour lesquels on peut identifier plusieurs variables explicatives. Ce document se concentre sur une classe particulière de ces algorithmes : les réseaux de neurones artificiels. Les réseaux de neurones tirent leur puissance de modélisation de leur capacité à capter les dépendances de haut niveau, c'est-à-dire qui impliquent plusieurs variables à la fois.

L'algorithme du réseau de neurone est donc un algorithme d'apprentissage, mettant en œuvre le principe de l'induction (c.a.d l'apprentissage par l'expérience). Grâce a leur capacité de classification et de généralisation (le fait d'être capable, par les exemples appris, de traiter des exemples distincts, encore non rencontrés, mais similaires), les réseaux de neurones sont le plus souvent utilisés dans des problèmes de nature statistique (analyse de données / prévision / classification) mais aussi robotique (contrôle et guidage de robots ou de véhicules autonomes), d'imagerie(reconnaissance des forme), et de traitement de signal ou encore de simulation d'apprentissage.

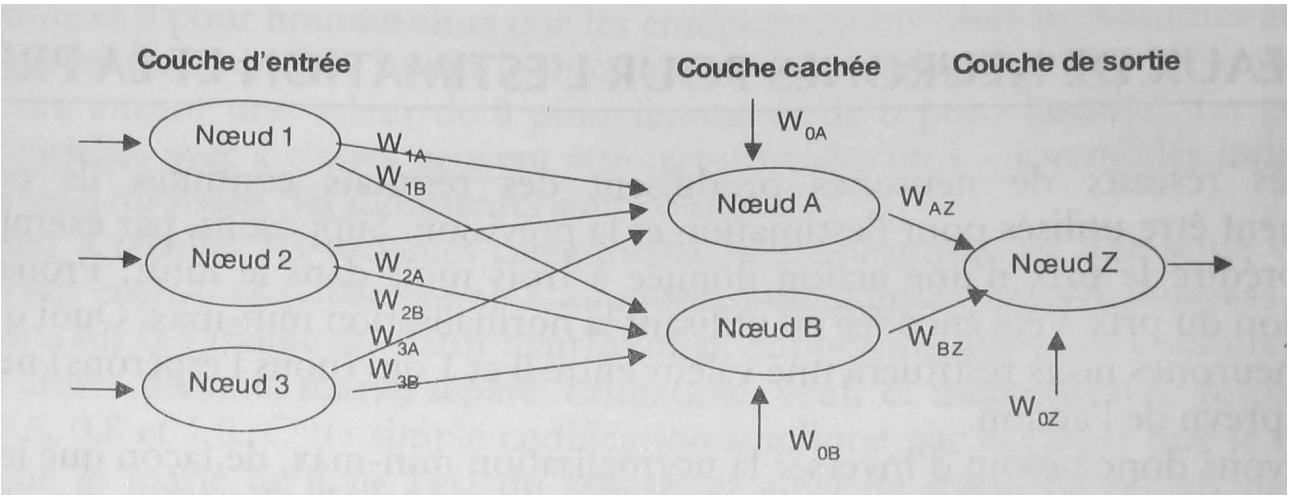
## *2. Fonctionnement d'un réseau neuronal :*

Un réseau neuronal est une modélisation mathématique du cerveau humain, il s'inspire donc du fonctionnement des neurones biologiques et prend corps dans un ordinateur sous forme d'algorithme. Le réseau neuronal peut se modifier lui-même en fonction des résultats de ses actions, ce qui permet l'apprentissage et la résolution de problèmes sans algorithme, donc sans programmation classique.

## *3. Structure d'un réseau de neurones:*

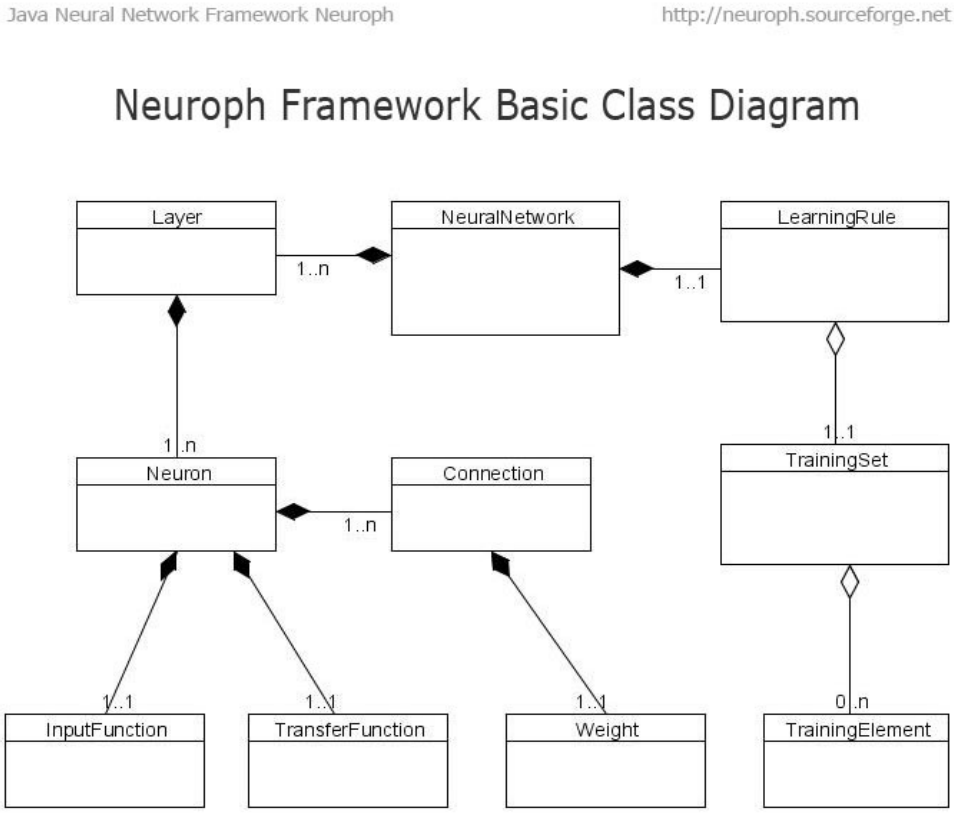
- Un réseau de neurones formels est disposé en couches de neurones formels.
- Les neurones sont appelés « nœuds ».
- La plupart des réseaux sont constitués de 3 couches successives : une **couche d'entrée**,une **couche cachée** et une **couche de sortie**. Toutefois, il peut y avoir 0 ou N couches cachées.
- D'une couche à l'autre, tous les nœuds de la première couche (nœuds « in ») sont reliés à tous les nœuds de la seconde (nœuds « out »).
- Chaque liaison a un poids : une valeur entre 0 et 1.
- Chaque nœuds des couches cachées et de sortie possède aussi un poids : une valeur entre 0 et 1.
- Le nombre de nœuds de la couche d'entrée dépend du nombre de variables prises en compte et de leur type. En simplifiant, on peut dire qu'on a un nœuds par variable en entrée.
- Le nombre de couches cachées et le nombre de nœuds pour chaque couche cachée est paramétrable par l'utilisateur.
- En général, la couche de sortie ne contient qu'un nœuds. Toutefois, elle peut en contenir plus. On peut dire que ce nœuds correspond à la variable de sortie.

### 1. Exemple d'un petit réseau de neurones



## *4. Choix du framework*

Nous avons choisi comme framework neuroph pour sa facilité de prise en main et pour la précision de sa documentation et pour sa flexibilité.




La classe **NeuralNetwork** est la classe de base du réseau de neurone, elle contient la structure principale et fonctionnalité utilisé dans un réseau de neurone. Contient l'ensemble des neurones(Layer), ainsi que les règles d'apprentissage(LearningRule).

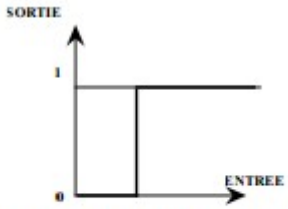
La classe **Layer** contient l'ensemble des neurones formant le réseau de neurones et permet d'accéder à l'ensemble des méthodes pour manipuler les neurones (add, remove, get, set, calculate, randomize).

La classe **Neuron** est l'élément contenant l'ensemble des données relatives à un neurone, c'est a dire les différentes liaisons et poids qui en résultent (Connection) ainsi que les différentes valeurs d'entrées(InputFonction) et fonctions de transfert(TransferFunction).

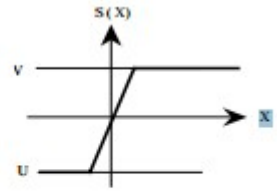
La classe **InputFunction** permanent est la classe représentant la couche d'entrée et permettra donc d'introduire les différentes variables prisent en compte. Elle contient la méthode getOutput qui renvoie l'ensemble des opérations (summingFunction) effectuées sur les variables d'entrées mises en parallèle avec le poids et un nœud voisin(weightsFunction). Sachant que les poids et opérations peuvent être différents selon le résultat souhaité.

La classe **Connection** crée un pont et spécifie le poids entre deux neurones.

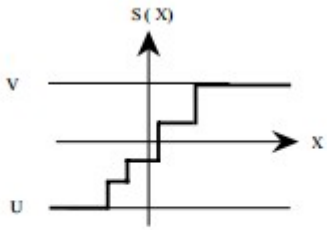
La classe **TransfertFonction** permet de transformer les entrées du réseau de neurone en sortie, selon des règles précises. Plusieurs méthodes d'équation existent pour réaliser une fonction de transfert, voici quelques exemples :



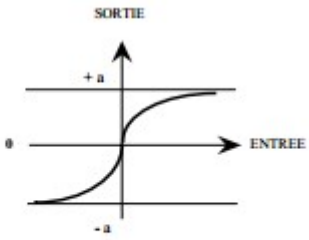

**a°)** Fonction de Heaviside, utilisée par McCulloch et Pitts.




**b°)** fonction linéaire à seuil.




**c°)** fonction linéaire multi-seuils.



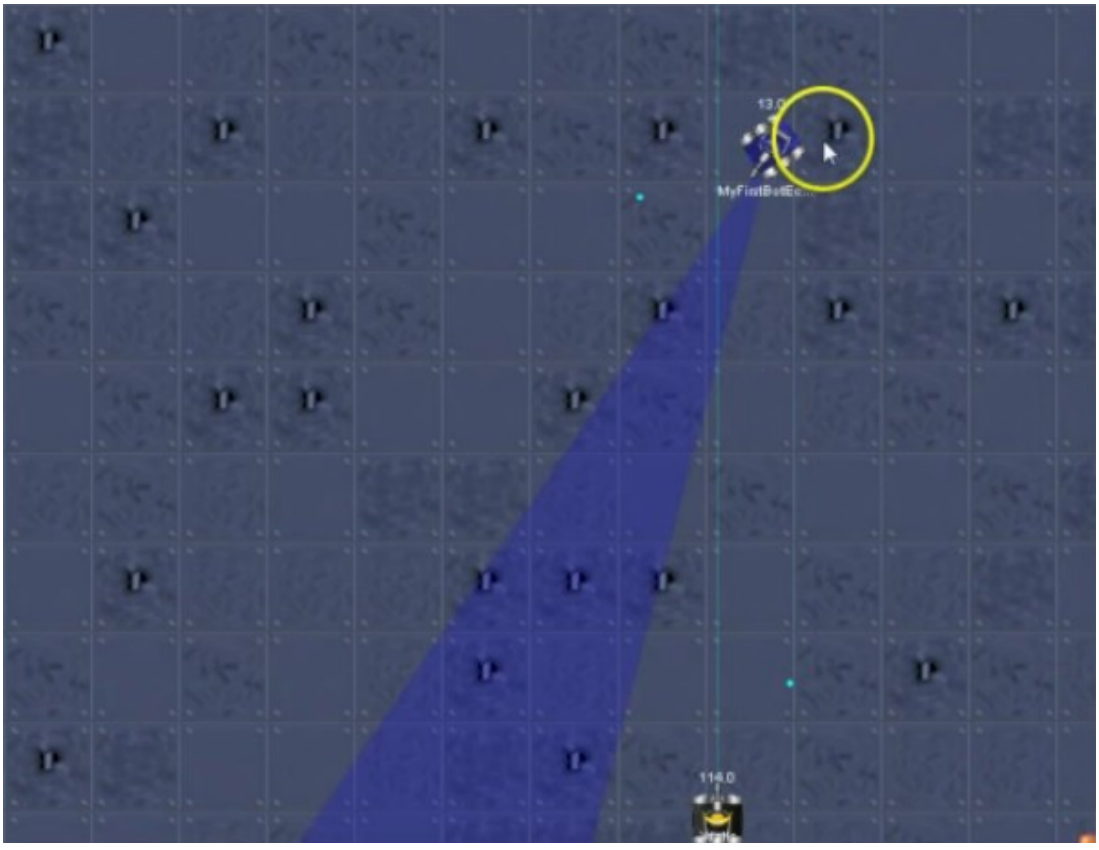

**d°)** fonction sigmoïde.

La classe **LearningRule** regroupe l'ensemble des algorithmes d'apprentissage et fournit les éléments principaux afin d'entraîner le réseau de neurones.

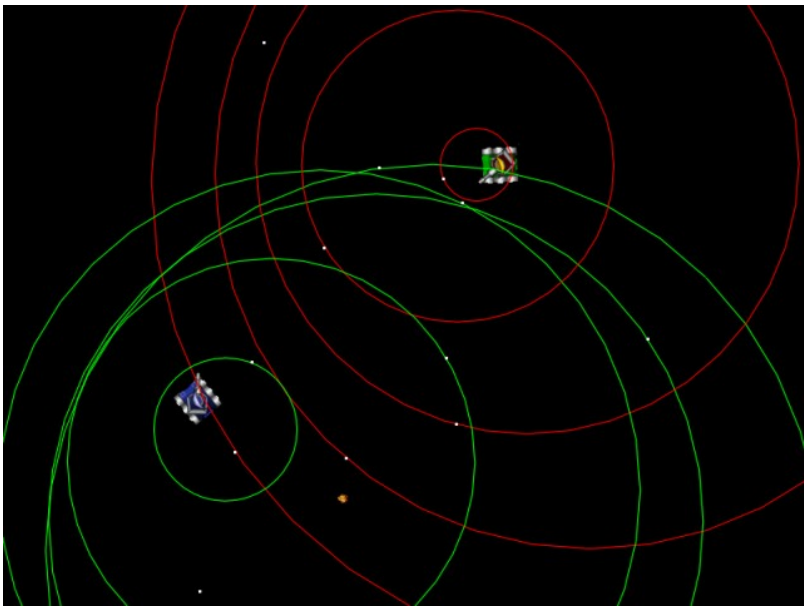
## *5. Utilisation*

Le but principale étant d'optimiser le déplacement du robot afin d'augmenter ses chances d'éviter les tirs ennemis, nous avons décidé de recueillir un maximum de données tests pour augmenter l'efficacité de l'algorithme de réseau de neurones. Nous avons également remarqué qu'il y existe plusieurs choix stratégiques possibles pour éviter les tirs ennemis, nous citerons les plus importants.

1. **Dive protection**: Le but de cette stratégie est d'empêcher votre robot d'approcher le robot ennemi avec un angle trop aigu, afin d'éviter d'être une cible facile. Dans un second temps cette stratégie fait en sorte que votre robot ne se place pas trop près de l'ennemi, causant collisions accidentel, ou rendant très difficile voir impossible l'esquive des balles .



2. **Wave Surfing**: Technique utilisée pour esquiver les tirs ennemis grâce aux étapes suivante:



- Détecter une baisse d'énergie pour savoir qu'une balle a été tirée à un endroit précis, puis créer une vague (wave) correspondante.
- Recueillir des données auprès des méthodes onHitByBullet ou onBulletHitBullet, en vérifiant qu'elles concernent bien la vague qui nous intéresse, ces données nous apprendrons les angles de tir du canon ennemi .
- Pour la balle la plus proche , utiliser la technique 'precise prediction ' pour déduire les zones de la vague que votre robot pourrait atteindre.
- Precise Prediction: En utilisant la physique de Robocode on pourra prédire avec précision la position de votre robot, pour déterminer où vous serez quand l'onde vous atteindra. Ensuite, le mécanisme de décision doit permettre un déplacement en conséquence.
- Se déplacer vers l'endroit le plus sûr et le plus facilement accessible à chaque vague.

# V. Algorithme génétique

L'Algorithme Génétique a été inventé par John Holland à l'Université du Michigan dans les années 70. Cet Algorithme fait parti des concepts se calquant sur le modèle naturel afin d'apporter des solutions à un problème donné. Ainsi, ce dernier s'inspire de la théorie de l'évolution selon Darwin, dans le but de trouver, non pas la solution parfaite, mais un ensemble de solutions les plus optimales possibles.

## *1. Principes*

Avant toute chose, il est important de définir quelques termes, empruntés au monde des biologistes, afin d'illustrer au mieux ces principes.

- Une population sera un ensemble d'individus
- Un individu sera une solution à un problème donné
- Un gène sera une partie d'une solution
- Une génération sera une itération de l'algorithme

L'algorithme aura la tâche de faire évoluer une population, avec pour objectif, d'en améliorer ses individus. Génération après génération, c'est donc un ensemble d'individus qui sera mis en avant, et bien que différents, ils seront de qualité équivalente.

De manière simple, le processus se découpera en cinq étapes :

1. Création de la population initiale
2. Évaluation de la population
3. Création de nouveaux individus
4. Insertion des nouveaux individus
5. Réitération du processus à partir du point 2.

### a) Création de la population initiale

Tout algorithme génétique a besoin d'une population à faire évoluer, le point de départ sera donc la création de la population initiale. Il n'y a pas de règles définies pour la création de cette dernière, mise à part le fait que chaque individu créé doit respecter la forme d'une solution potentielle.

Ici, le but n'est pas de créer des individus viables, mais bien de remplir les données initiales à manipuler. Il sera donc très intéressant d'effectuer une création aléatoire dans le but d'obtenir une certaine diversité qui augmentera nos chances de trouver des solutions optimales.

Enfin, dernier point essentiel, la taille de la population. De la même manière, il n'existe pas de règle définie, et celle-ci sera le plus souvent un compromis entre temps d'exécution et qualité des solutions trouvées.

### b) Évaluation de la population

L'évaluation est une étape importante du processus, c'est à ce moment que nous allons pouvoir mettre en avant des individus par rapport à d'autres, en leur attribuant une note, ou toute autre méthode de classement.

Ici aussi, la méthode dépend du problème que l'on cherche à résoudre et s'avérera plus ou moins évidente. Par exemple, pour une solution mono-critère numérique, il sera aisé de comparer deux individus alors qu'une solution multicritères mettra en scène la notion de dominance d'individus.

### c) Création de nouveaux individus

C'est ici que va se passer toute l'évolution de la population qui est décomposée en plusieurs étapes: la sélection, le croisement et la mutation.

- La sélection

La première étape va nous permettre de choisir quels individus vont participer à la création de ces nouveaux individus. Encore une fois le choix du nombre est libre bien qu'il ne soit pas recommandé de modifier tous les individus.

Il est possible de sélectionner au hasard les participants ou bien d'utiliser d'autres méthodes comme la roulette, la sélection par rang, par tournoi ou encore l'élitisme.

- Le croisement

Après la sélection vient l'évolution. La première méthode permet de simuler des reproductions d'individus dans le but d'en créer de nouveaux. Elle consiste en la découpe des individus en N morceaux et la construction de nouvelles solutions avec les gènes obtenus. Il est donc possible de créer M nouveaux individus avec M sélectionnés ou plus si l'on décide de réutiliser les gènes plusieurs fois.

- La mutation

Cette solution nous demandera de modifier les individus existants en modifiant un ou plusieurs gènes. En effet, une modification mineure peut très bien changer considérablement l'évaluation d'un individu.

Cependant, l'évolution est soumise à une seule contrainte : chaque nouvel individu doit être de la forme d'une solution potentielle. De plus, il est impossible de savoir si les nouvelles solutions seront meilleures ou moins bonnes, ces étapes sont là afin d'apporter de nouvelles possibilités dans le but d'arriver à de meilleures solutions.

### d) Insertion des nouveaux individus

Une fois l'évolution terminée, il est temps de sélectionner les individus de la prochaine génération. Une nouvelle fois, cette étape de sélection dépendra des traitements effectués précédemment et est libre, bien qu'il soit conseillé de garder une taille de population constante. Une manière efficace consistera à insérer les nouveaux individus dans la population, la réévaluer, puis ne garder que les N meilleurs.

### e) Réitération du processus

Il n'existe pas de nombre de générations prédéfini, il faudra ajuster le nombre de traitements en fonction du problème donnée et des solutions obtenues. Il n'est généralement pas possible de trouver des solutions convenables avec peu de générations et celles-ci finiront par ne plus évoluer avec un nombre trop important.

Une fois le nombre de générations atteint, on se retrouve avec les solutions possibles se rapprochant des solutions optimales.

## 2. Choix du framework

Le choix du framework s'est porté sur JGAP, une bibliothèque centré sur les algorithmes génétiques écrite en JAVA. Son atout majeur est d'avoir été modélisé de façon à être simplement et rapidement utilisable tout en restant très modulaire.

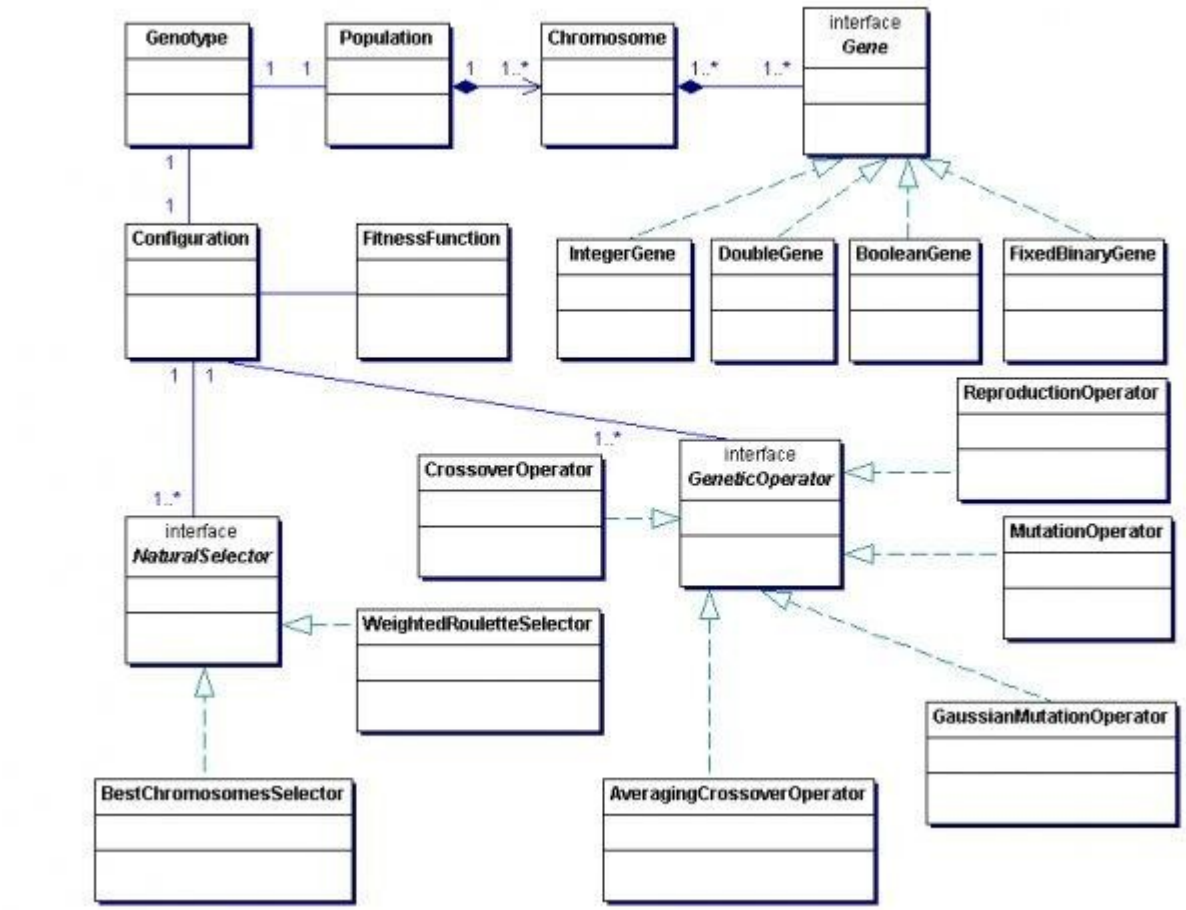



*Illustration 1: Diagramme des classes les plus importantes de JGAP*

Comme on peut le voir sur sur l'illustration précédente, toutes les étapes et représentations nécessaires au bon fonctionnement de l'algorithme sont présentes. Il est d'ailleurs possible d'utiliser le framework très rapidement en ayant à renseigner, pour les différentes opérations, uniquement la fonction d'évaluation, cf. la classe ***FitnessFunction***.

De plus, pour une utilisation plus poussée ou plus adaptée, il sera intéressant de pouvoir implémenter une partie ou l'ensemble des opérations grâce aux interfaces ***GeneticOperator*** et ***NaturalSelector*** de la même manière que la représentation des individus, dits ***Chromosome***, et de leurs gènes.

## 3. Utilisation

Dans l'objectif d'optimiser la fonctionnalité de tir, nous sommes arrivé à la stratégie suivante.

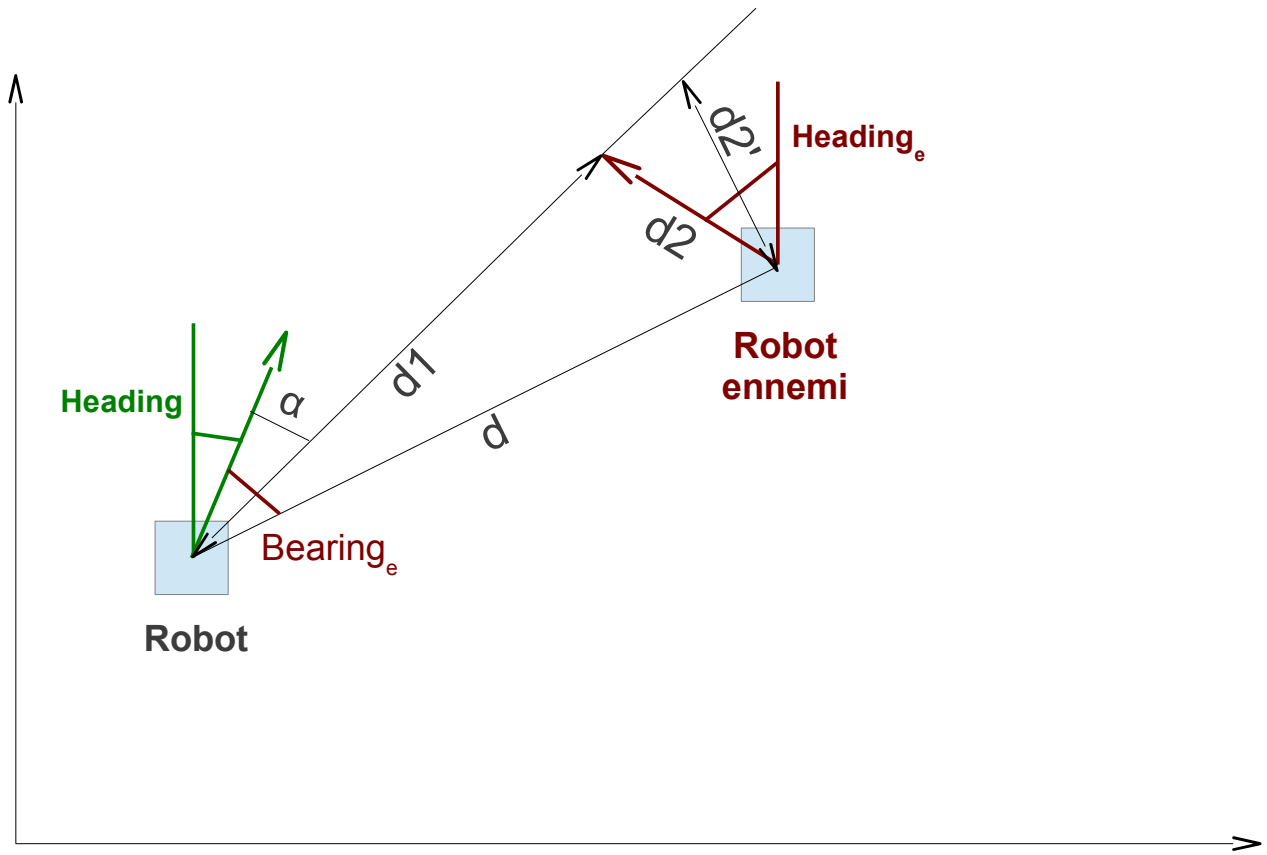



*Illustration 2 : Représentation des données nécessaires lors de la détection d'un ennemi*

**Formules utiles :**

- BulletSpeed = 20 – 3 * power
- $LateralVelocity_E = Velocity_E * Sin(Heading_E – (Bearing_E + Heading))$

**Objectif :**

Soit **d1** la distance parcouru par la balle durant la période **T1** et **d2** la distance parcouru par le robot ennemi durant la période **T2**. L'objectif est de déterminer l'angle de tir **α** tel que **T1** = **T2** avec une puissance de tir '**power**' inversement proportionnelle à la distance **d** séparant les deux robots. Ainsi, on privilégie des dégâts importants lorsque l'ennemi est proche, au détriment de la vitesse de tir, et une probabilité plus importante de toucher l'ennemi, en l'occurrence qu'il garde la même direction, lorsque celui est loin en augmentant la vitesse au détriment des dégâts.

**Représentation :**

Chaque individu représentera une solution de la forme **{ α, power }** dans le but d'optimiser l'angle et la puissance de tir en fonction de la position relative de l'ennemi par rapport à notre robot, de sa distance ainsi que sa vitesse latérale dans le but d'analyser ses mouvements et pourvoir déterminer sa prochaine position possible.

**Évaluation :**

A ce jour, la méthode d'évaluation des solutions est la suivante :

- Evaluation de la puissance en fonction de la distance entre l'adversaire et notre robot.En effet, on va valoriser une puissance forte si l'ennemi est proche. A l'inverse, on valorisera une faible puissance pour un ennemi plus éloigné. Nous déterminerons les paliers de distance lors des premiers tests.
- Evaluation de l'angle de tir en fonction de la position actuelle de l'adversaire et sa position estimée. En effet, l'angle **α** de tire devra être entre la position actuelle de l'adversaire et sa position estimée par rapport à notre robot qui est la source, on l'appellera « delta ».

  alpha <= |delta|

## VI. Conclusion

De part l'étude des deux algorithmes d'intelligence artificielle (RN et AG), nous avons établi deux stratégies d'études, dans le but créer ou d'optimiser un robot. Le Réseau de Neurones se concentrera sur le mouvement et l'esquive de projectiles, alors que l'Algorithme Génétique se penchera sur l'optimisation du système de visée du robot.

De manière générale, nous n'avons pu recueillir d'exemples concrets à étudier, les robots déjà existant, utilisant les deux types d'algorithme, ne les appliquaient pas dans les domaines respectifs que nous avons choisis. L'étude du Réseau de Neurones fut laborieuse, de par sa nature très mathématique et théorique. Cependant grâce a l'abondance de documentation traitant le sujet, l'essentiel des notions et du fonctionnement a été assimilé.

Enfin, la prochaine étape du projet consistera à implémenter et tester les stratégies établies. Il possible que l'on s'affronte entre nous et/ou de créer un robot unique alliant les deux stratégies.

## Remerciements

**Ludovic Bonnefoy**, tuteur, encadrent, et à l'origine du projet de M1/M2.

**Borris Deltienne** et **Sophie Nabitz** responsables des projets M1/M2.

## VII. Calendrier

Equipe : 4 personnes
Taux horraire/personnes : 180 heures

- Tests API Robocode 10 heures
- Tests frameworks IA 10 heures
- Modélisation 30 heures
- Implementations des frameworks et des stratégies 45 heures
- Tests et optimisations des stratégies 30 heures
- Debugging 25 heures
- Test implementation et test des deux stratégies en un seul robot 15 heures
- Redaction rapport et préparation de la soutenance 15 heures

# VIII. Annexes

Site officiel du robocode:

http://robocode.sourceforge.net/

Explication de l'Algorithme de Réseau de Neurones :

http://alp.developpez.com/tutoriels/intelligence-artificielle/reseaux-de-neurones/

http://www.lrde.epita.fr/~sigoure/cours_ReseauxNeurones.pdf

http://bliaudet.free.fr/IMG/pdf/Cours_de_data_mining_8-Reseaux_de_neurones-EPF.pdf

Documentation de Neuroph:

http://neuroph.sourceforge.net/javadoc/index.html

Wikipedia de l'Algorithme Génétique:

http://fr.wikipedia.org/wiki/Algorithme_g%C3%A9n%C3%A9tique

Framework JAVA sur l'algorithme génétique:

http://jgap.sourceforge.net/

Explication de l'Algorithme Génétique:
http://khayyam.developpez.com/articles/algo/genetic/

Exemple d'utilisation de l'Algorithme Génétique :

http://khayyam.developpez.com/articles/algo/voyageur-de-commerce/genetique/

Sean Luke, 2009, Essentials of Metaheuristics, Lulu, avaible at
http://cs.gmu.edu/~sean/book/metaheuristics/

Théorie de l'évolution selon Darwin:

http://fr.wikipedia.org/wiki/Charles_Darwin#D.C3.A9but_de_la_th.C3.A9orie_de_l.27.C3.A9volution_de_Darwin