Projet Robocode

Encadré par : **Ludovic Bonnefoy**

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

Projet Robocode

I. Introduction

Robocode est un jeu vidéo qui a été distribué par IBM, à des fins éducatives. En effet, son but principal 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.

Rappelons que les deux hypothèses de travail étaient :

- établir une IA qui permet d'esquiver les projectiles des robots adverses. Pour cela, un groupe à utilisé un algorithme de réseau de neurones.
- élaborer une autre IA qui permet de tirer sur l'adversaire. L'autre groupe a utilisé un algorithme génétique.

Nous allons présenter en détails le travail et les observations de chaque groupe, avant de donner une analyse globale quant à la pertinence d'utiliser de tels algorithmes dans ce domaine précis d'application.

II. Réseau de neurones

Il est ici question d'utiliser un réseau de neurones afin d'optimiser le déplacement du robot et ainsi esquiver les tirs ennemis. Cet algorithme nous permet à travers l'apprentissage par l'expérience de créer des règles permettant un résultat optimal. Nous allons donc vous exposer les différentes prises de décisions tout au long du projet, les difficultés que nous avons dû surmonter, mais aussi l'ensemble de la phase d'implémentation de l'algorithme lié à l'API robocode.

1. Cheminement

Dans la première partie du projet, lors du précédent semestre, nous avons dû nous imprégner de l'ensemble des outils fournis par l'Api Robocode. Mais la majeure partie de l'étude que nous avons effectué se portait sur le réseau de neurones. En effet, la logique et les concepts liés à celui-ci ne sont pas forcément simple à cerner. L'enjeu était donc de trouver comment lier le mouvement du robot au réseau de neurones. Nous avons choisi un framework propre à celui-ci, Neuroph, ce choix se justifie par la pertinence de sa documentation mais également car il nous propose une visualisation graphique du réseau de neurones. Pour cela, il a fallut chercher l'ensemble des variables étant susceptible d'influer sur l'esquive et le déplacement du robot. Pour cela, nous nous sommes inspiré de la technique de WaveSurfing qui est une méthode d'esquive déterministe extrêmement efficace. Cette technique nous a donc permis de rendre l'ensemble des données fournis par l'API Robocode plus concrète et de nous familiariser avec son utilisation. C'est donc pour cela que nous avons commencé par implémenter l'ensemble de cette technique.

La grande difficulté a été de déterminer les valeurs de sortie du réseau de neurones pouvant influer efficacement sur le comportement voulu (esquiver les tirs ennemis). Celles-ci se sont vues être modifiées tout au long de nos recherches et de la réalisation de notre IA. Nous avons choisi dans un premier temps d'utiliser en sortie la position optimale afin d'éviter le projectile. Cependant cela implique donc que l'on ai déjà implémenté un algorithme aillant un comportement irréprochable. La décision s'est donc faite au moment où nous nous sommes lancés dans le développement de l'API.

Afin d'avoir un exemple homogène sur lequel le réseau de neurones pourrait s'entraîner, nous avons utiliser une autre technique sensiblement différente du WaveSurfing puisque ce dernier était basé sur le principe de « *stop and go* » et sur un mode de déplacement aléatoire.

Nous avons ainsi parcouru succinctement les différents objets d'étude qui ont permis l'élaboration du projet. C'est pour cela que nous allons maintenant aborder la partie technique de ce dernier.

2. Outils, technologies et structure de données

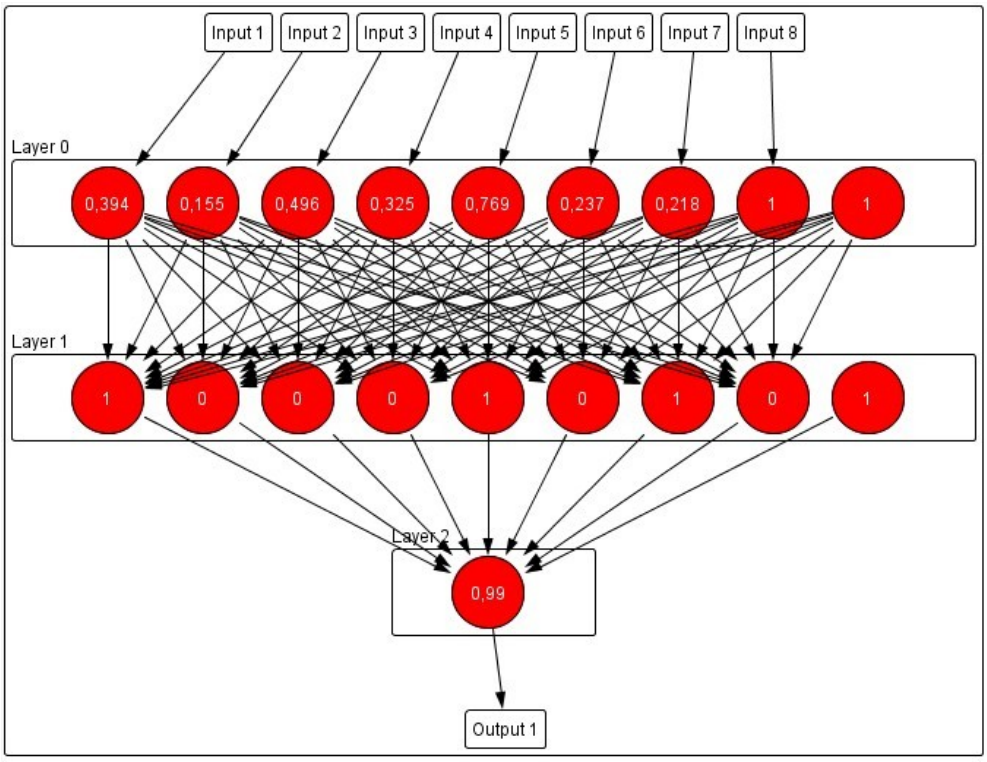
Le réseau de neurones repose sur un principe d'apprentissage par l'expérience. Pour cela, nous nous sommes basés sur plusieurs phases de développement. Dans un premier temps, nous avons mis en place d'un nouveau robot qui génère la base d'apprentissage de notre réseau de neurones. Par la suite, nous allons vous exposer les différents choix et recherches effectués afin d'optimiser la phase d'entraînement grâce à NeurophStudio. Et enfin, l'implémentation de la base d'apprentissage à l'API robocode.

8 variables d'entrée :

- Postion x et y du tir ennemi (fireXLocation et fireYLocation)
- Position x et y de notre robot (myXPostion et myYPostion)
- Angle de tir (directAngle)
- Vitesse du tir (bulletVelocity)
- Distance parcourue (distanceTraveled)
- Direction du projectile (direction)

1 variable de sortie :

- Touché ou non (hitted)



a) Mise en place des données de base :

Nous avons donc dû choisir un technique de déplacement avec une efficacité intermédiaire afin d'avoir une bonne base de départ pour l'apprentissage. La technique que nous avons retenu, est un mélange entre le « *stop and go* » (attend que quelque chose se passe et bouge) et un déplacement aléatoir. Cette technique nous permet donc après plusieurs rounds, d'avoir parcourus un maximum de positions et d'avoir alterné esquives réussies et manquées. Pour sauvegarder les résultat, nous avons donc fait en sorte d'enregistrer les variables d'entrées dans un fichier. Nous avons donc créé une classe Wave rassemblant toute les informations concernant le tir. Chaque Wave sera enregistrée dans une liste qui sera ensuite copiée dans un fichier de données. Pour cela nous avons donc utilisé un ensemble de méthodes fournis par l'API Robocode.

Voici la liste des différentes méthodes et du rôle de chacune d'elle et quelque spécifications :

- **run :** Lance le thread lié au robot et initialise certaines variables.

- **onScannedRobot :** Cette méthode est déclenchée lors de la détection du robot ennemi, elle permet de définir son comportement et ses différentes actions. Ainsi cette méthode va nous permettre de calculer et récupérer l'ensemble des données utiles à l'entraînement du réseau de neurones. Ainsi on définit également le comportement de notre robot (esquive et tir).

- **onHitWall :** Change de direction au contacte d'un mur.

- **onDeath :** Incrémente le nombre de fois où notre robot est mort.

- **bulletVelocity :** Retourne la vitesse de la balle à partir de sa puissance. (px/tours)

- **updateNeuralData :** Met à jour la liste des données pour le réseau de neurones, dans le cas où le tir ennemi manque notre robot avec comme valeur de sortie 1.

- **onBattleEnded :** Se déclanche à la fin du match. Appel la méthode **writeLog** qui permet l'enregistrement des données du match dans un fichier.

- **onBulletHit :** Met à jour la liste des données pour le réseau de neurones, dans le cas où le tir ennemi touche notre robot avec comme valeur de sortie 0.

- **project :** Projection de la position d'un robot par rapport à sa position, son angle, et la distance qu'il va parcourir.

- **Wave.collection :** Prend en argument un entier spécifiant si le tir a touché notre robot et renvoie l'ensemble des variables utiles au réseau de neurones sous forme de liste.

b) Training set (apprentissage)

Nous récupérons donc les données à la fin de la bataille et ce grâce à la méthode onBattleEnd() en les enregistrant dans un fichiers.

Nous avons fait ce choix pour tester différentes configurations de réseau de neurones à l'aide de NeurophStudio qui est une GUI basée sur Netbeans Plateform permettant la création de réseau de neurones de manière graphique, facilitant le suivi de l'évolution de l'apprentissage grâce à des graphes.

Normalisation des donnés :

Étant donné que les entrées ne sont pas toutes sur la même échelle, une normalisation de celles-ci s'impose. Pour ce faire, nous avons créé une classe *Normalize* qui va lire le fichier généré précédemment, puis le normaliser grâce à la formule suivante:

$$B = (A - \min(A)) / (\max(A) - \min(A)) * (D - C) + C$$

B est la valeur normalisée

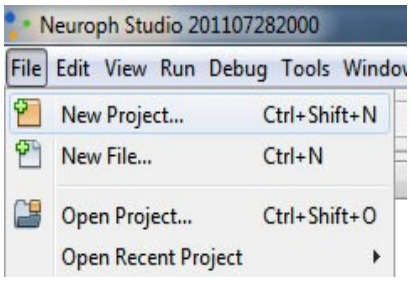
A la valeur à normaliser

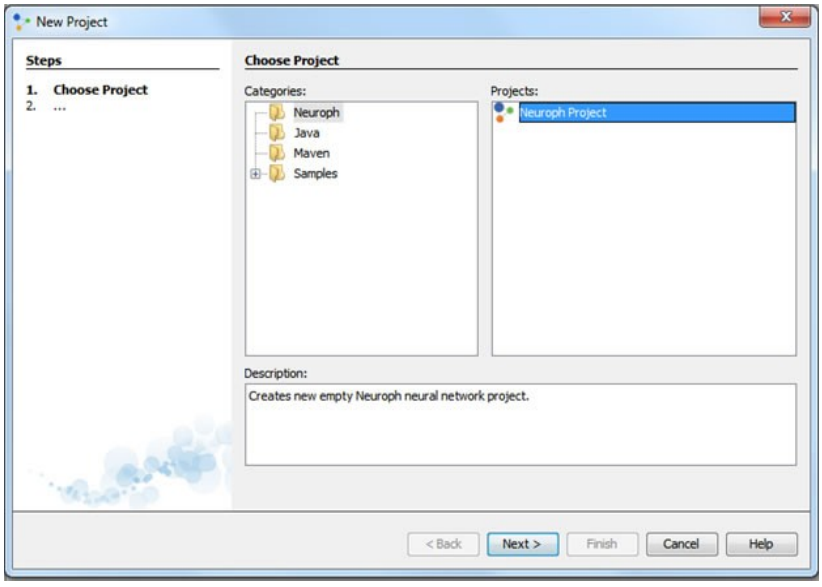
D et C sont les valeurs qui vont borner B, dans notre cas 0 <B< 1

Le résultat sera alors enregistré dans un nouveau fichier.

Création d'un projet Neuroph :

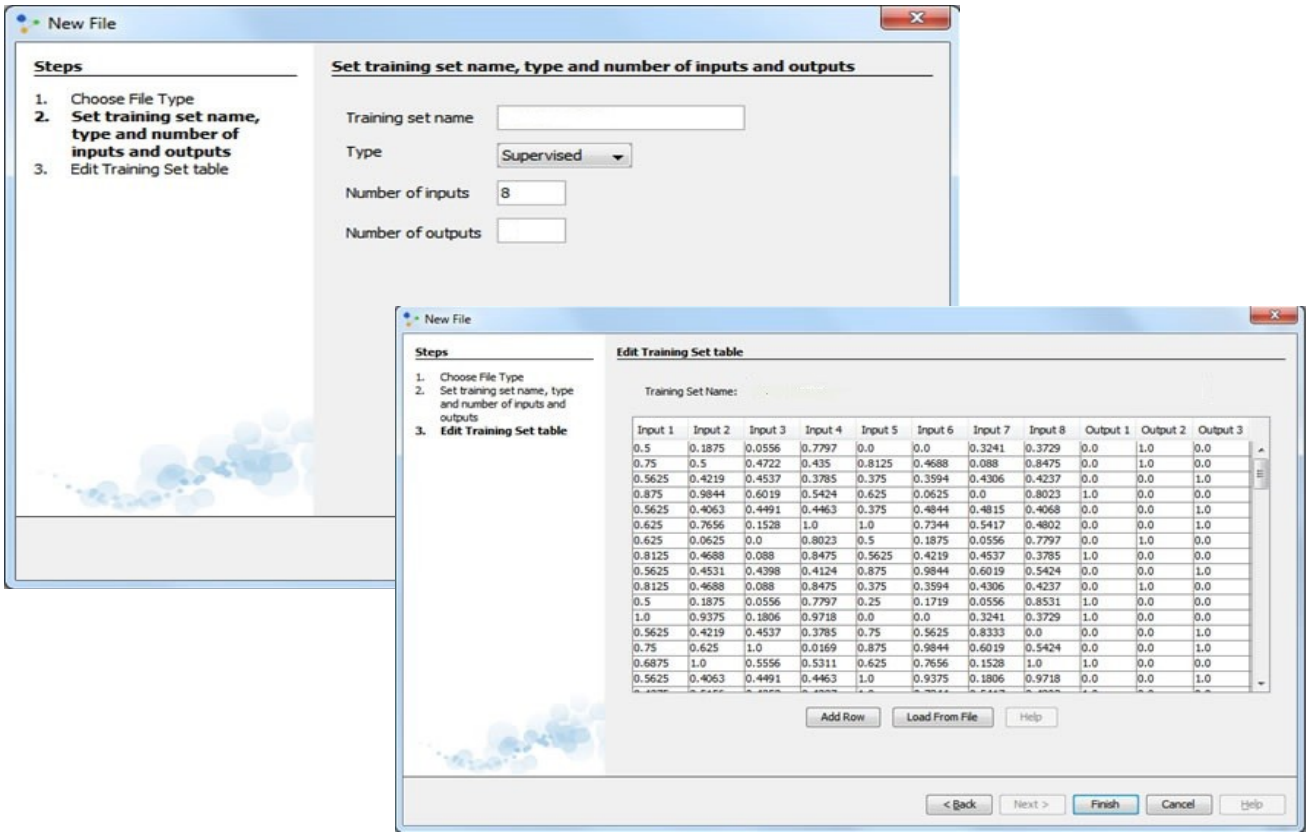
Ensuite, nous allons créer un projet Neuroph grâce à l'application NeurophStudio, de la même manière que l'on crée n'importe quel projet sur Netbeans.





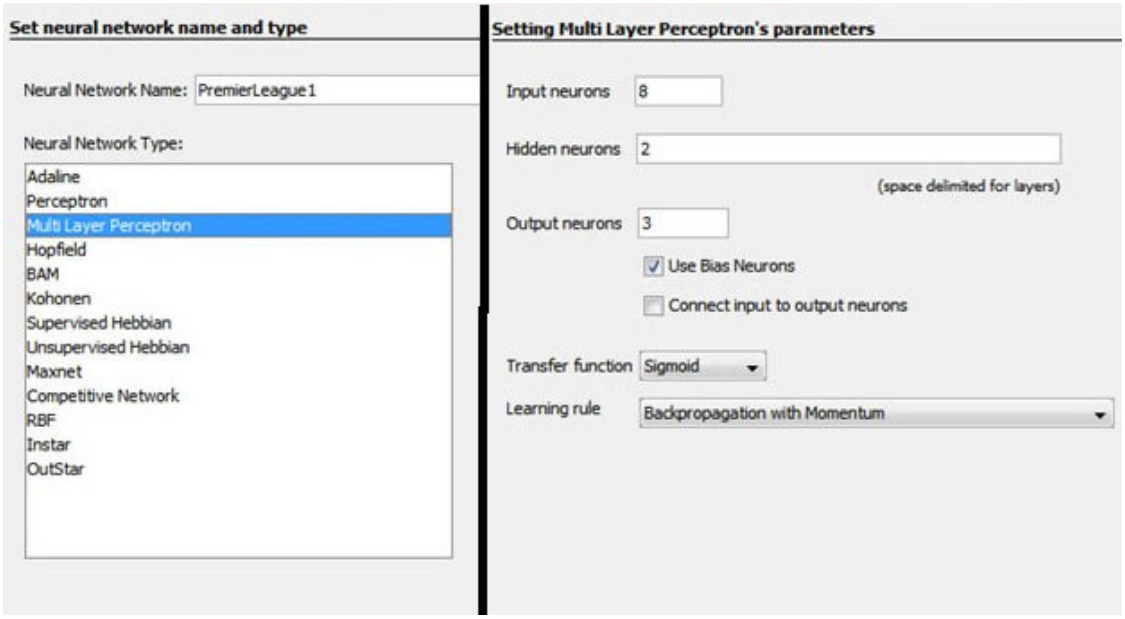
Création d'un ensemble d'apprentissage :

Après création de notre projet, nous avons créé un ensemble d'apprentissage à partir du fichier généré précédemment, après normalisation. Nous avons donc créé un fichier de type « *training set* » en entrant les données entrantes et sortantes pour le réseau de neurones. De plus, il est possible d'entrer les données une par une ou de charger les données à partir d'un fichier, nous évidemment avons choisi la seconde option.



Création d'un réseau de neurone

Dans un premier temps, une des étapes importantes dans la création d'un réseau de neurones est de définir l'ensemble des configurations. Le choix du type que nous avons effectué étant le Multi Layer Perceptron. Ce type de réseau de neurones est le plus populaire car il permet de modéliser des fonctions de complexités presque arbitraires. Le nombre de couches et de neurones par couches détermine sa complexité.



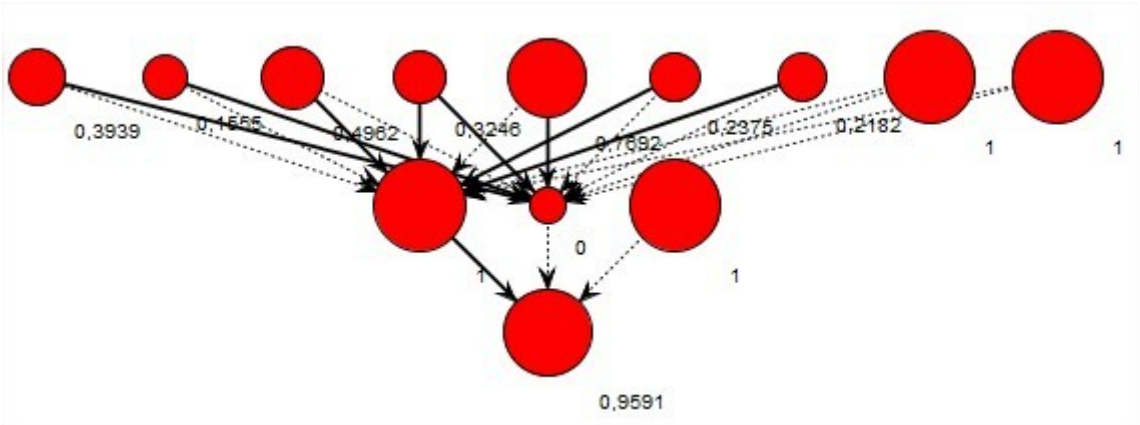
Nous avons testé différentes configurations de réseau de neurones en précisant le nombre d'entrés (8), mais surtout le nombre de neurones en couche cachée, pour la fonction de transfert. Nous avons choisi la fonction Sigmoïde f(x)=1/1+e^-x, ce choix nous a paru évident étant donné que nos donnés ont été normalisés de manière à être bornées entre 0 et 1. Une fonction de transfert entre dans le processus de calcul de poids d'un neurone et sert principalement à introduire une non-linéarité dans le fonctionnement du neurone.

Ainsi l'algorithme que nous avont retenu pour l'entrainement, est l'algorithme de backpropagation, il consiste à :

1. Comparaison de la sortie du réseau avec la sortie ciblée.
2. Calcul de l'erreur en sortie de chacun des neurones du réseau.
3. Calcul, pour chacun des neurones, de la valeur de sortie qui aurait été correcte.
4. Définition de l'augmentation ou de la diminution nécessaire pour obtenir cette valeur (erreur locale).
5. Ajustement du poids de chaque connexion vers l'erreur locale la plus faible.
6. Attribution d'un blâme à tous les neurones précédents.
7. Recommencer à partir de l'étape 4, sur les neurones précédent en utilisant le blâme comme erreur.

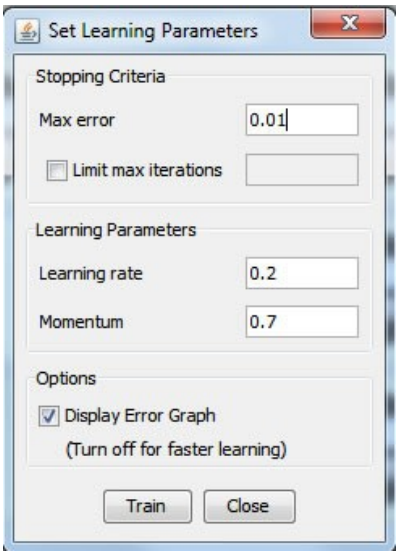
3. Problèmes et difficultés rencontrés

- ✗ Trouver les variables d'entrées du réseau de neurones parmi l'ensemble des données fournis par l'environnement.
  - Solution retenue : Récupération d'une partie des données utilisées dans le WaveSurfing.
- ✗ Se fixer une variable de sortie.
  - Solution retenue : Sortie binaire ; tir esquivé ou non.
- ✗ Sécurité Robocode rendant difficile l'écriture dans un fichier externe.
  - Solution retenue : Désactivation de cette sécurité dans le robocode.bat
- ✗ Malgré différents fichiers d'exemples et une tentative d'apprentissage avec de multiples configuration de réseau de neurones aucun résultat n'était satisfaisant. En effet nous ne somme jamais passé en dessous de la barre des 0,01 au Mean error lors des test.
  - Aucune solution trouvée.
- ✗ Import de la bibliothèque Neuroph au sein de l'API Robocode.
  - Solution retenue : Modification de l'exécutable robocode (.bat) en précisant le chemin vers la librairie neuroph.
- ✗ Choix du nombre de neurone dans la couche caché.
  - Solution retenue : Nous avons testé une multitude de configuration et avons retenue celle obtenant les meilleurs résultat.

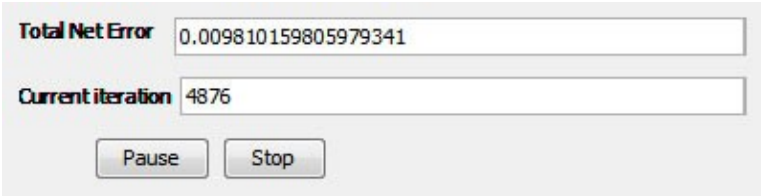


4. Résultats et analyses

- Test de multiples réseaux de neurones afin de trouver la configurations optimale :
  - configuration :8 entrées 8 neurones dans la couche cachée 1 sortie a été l'essai le plus concluant. L'entrainement a réussi avec une erreur inférieur à l'erreur max fixé (0,01), mais les résultats de test n'ont pas étés très concluant. Comptant un certain nombre d'erreurs individuelles beaucoup trop élevées.



  - Sachant que notre meilleur résultat reste au dessus de l'erreur fixée (0,0164834....).




```
Input: 0,1721; 0,7465; 0,652; 0,1111; 0,7692; 0,4251; 0,6; 1;  Output: 0,987; Desired output: 1;  Error: -0,013;
Input: 0,2087; 0,7893; 0,6245; 0,004; 0,7692; 0,447; 0,6364; 1;  Output: 0,9908; Desired output: 1;  Error: -0,0092;
Input: 0,2489; 0,8258; 0,6238; 0,0016; 0,7692; 0,4548; 0,6364; 1;  Output: 0,9911; Desired output: 1;  Error: -0,0089;
Input: 0,4748; 0,1268; 0,4468; 0,3741; 0,7692; 0,1559; 0,1818; 1;  Output: 0,9753; Desired output: 1;  Error: -0,0247;
Input: 0,3939; 0,1555; 0,4962; 0,3246; 0,7692; 0,2375; 0,2182; 1;  Output: 0,998; Desired output: 1;  Error: -0,002;
Total Mean Square Error: 0.021436616445403123
```

Le résultat de l'entrainement génère un fichier qui sera chargé lors du démarage du robot, grâce à une méthode de l'API Neuroph. De cette méthode, un objet de type neuralNetwork sera créé. Il suffira de mettre de nouveaux cas constatés en entrée pour déterminer la solution optimale. De plus, le réseau s'entraînnera avec les nouveaux paramètres établies.

III. Algorithme génétique

Dans cette partie, il est question d'optimiser le système de tir à l'aide d'un algorithme génétique. Tout d'abord, nous détaillerons les différentes optiques que nous avons suivi lors du développement. Ensuite, nous allons présenter les différents outils et structures de données utilisés lors du développement. Enfin, nous énoncerons les problèmes que nous avons rencontré lors du développement.

1. Cheminement

Suite à la phase d'analyse du semestre 1, nous avions décidé de trouver une façon d'optimiser l'angle de visé du canon, dans le but d'anticiper au mieux les mouvements de l'adversaire, mais aussi d'adapter la puissance du tir en fonction de la distance séparant les deux concurrents. Dans cette lancée, nous avons d'abord commencé par implémenter cette solution avant de nous apercevoir, suite aux tests et aux remarques de Boris DETIENNE, qu'il était possible de déterminer l'angle de tir de manière plus précise et moins lourde en ressource. En effet, l'algorithme génétique ne nous renvoie qu'une approximation d'une solution optimale alors que dans ce cas, un calcul déterministe se révèle plus rapide et performant.

De ce fait, nous avons décidé de changer d'optique. Nous sommes partis sur l'idée de maximiser les dégats infligés à l'adversaire en fonction de la distance à laquelle ce dernier se trouve. Nous avons désormais une solution représentée par un ensemble de paliers mettant en lien la puissance de tir ainsi que la distance à laquelle se trouve l'adversaire. Nous avons donc établi un rapport quantitatif qui permet d'apprécier la puissance du projectile et la distance en fonction du temps.

Le rapport suivant : P*V/D, avec P les dégâts infligés, V la vitesse du projectile et D la distance entre les deux robots, permet d'évaluer l'efficience de chaque palier de tir de notre robot en comparant leur dégâts effectifs par tour. La qualité d'une solution sera ici la somme des rapports de ses paliers.

Nous la partie suivante, nous détaillerons les moyens utilisés ainsi que les structures de données mises en place afin de rendre cette solution fonctionnelle.

2. Outils, technologies et structure de données

Pour le développement de ce robot, nous avons donc opté pour le langage Java avec l'API Robocode associée, couplé à la bibliothèque d'algorithme génétique pour Java JGAP. Pour rappel, 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. Nous nous sommes servi de l'IDE Eclipse pour le développement et le débogage de l'application.

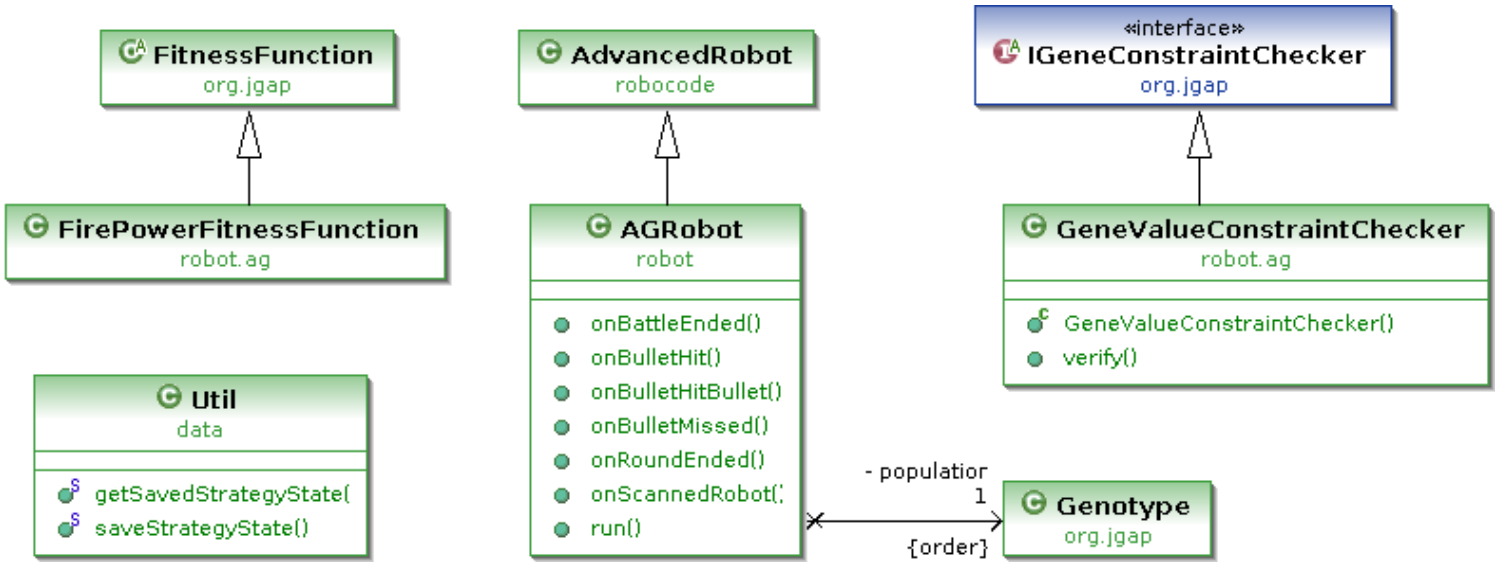


Illustration 1: Diagramme simplifié de la mise en place de l'algorithme génétique

La classe *AGRobot* est la classe principale lancée par l'application Robocode. Sa méthode principale, *run()*, exécute les instructions suivantes :

- Initialisation de l'algorithme génétique uniquement lors du premier round
  - Initialisation de la configuration de JGAP
  - Initialisation du chromosome (solution) de référence
  - Génération ou restauration du génotype (population de chromosomes)
- Récupération de la meilleure solution
- Boucle principale
  - Instructions de déplacements aléatoires

La méthode *onScannedRobot()* contient toute la logique de la stratégie.

- Calcul de l'angle de tir en fonction de la puissance associée au palier approprié de la meilleure solution actuelle
- Mouvement du canon grâce à l'angle calculé précédemment
- Tir du projectile avec la puissance adéquate

La méthode *onRoundEnded()* sert principalement à faire évoluer la population de chromosomes entre chaque manche.

Enfin, la méthode *onBattleEnded()* nous permet de sauvegarder l'état du génotype dans un fichier xml entre chaque match grâce à la classe *Util*.

La classe *GeneValueConstraintChecker* a pour rôle de valider un gène donné avant son insertion dans un chromosome. Cette dernière, constituée d'une unique méthode *verify*, nous permet de vérifier que la valeur de la puissance, ou la distance, du palier que l'on souhaite modifier soit bien comprise entre celle du palier précédent et du palier suivant.

Pour finir, la classe *FirePowerFitnessFunction* a pour rôle d'évaluer un chromosome donné. C'est ici que nous allons calculer le rapport « P*V/D » pour chaque palier de la solution puis d'en faire la somme afin de classer le chromosome au sein de la population.

Déroulement d'une manche en adéquation avec notre stratégie :

- Initialisation de l'algorithme génétique uniquement lors du premier round
- Récupération de la meilleure solution
- Boucle principale
  - Instructions de déplacements aléatoires

- Lors de la récupération de l'événement *ScannedRobot*
  - Calcul de langle de tir
  - Mouvement du canon
  - Tir
- En fin de manche
  - Evolution de la population de solutions

3. Problèmes et difficultés rencontrés

- **Calcul de l'angle :**

  Nous avons eu particulièrement du mal à calculer l'angle que doit avoir la tourelle pour pouvoir tirer sur l'adversaire. En effet, le référentiel trigonométrique sur lequel s'appuit l'API robocode n'est pas le même que celui en mathématique. Il était donc difficile de calculer correctement le résultat.

- **Problème de sauvegarde et de chargement :**

  Nous avons rencontré un problème au niveau de la sauvegarde/chargement de nos données sous format XML, proposé par le XMLManager du framework JGAP. En effet, on ne pouvait pas relancer directement le programme (RESTART). On était obligé de fermer ce dernier et de relancer un nouveau processus, sinon une exception était levée. Pour le chargement, on a une erreur au niveau du framework, au niveau du DOM XML.

- **Obliger de mettre la sécurité off sur robocode :**

  Nous avons dû désactiver le Security Manager de l'application robocode, sinon notre AGrobot ne pouvait pas tourner sur l'application.

- **Utilisation, manque et difficulté de configuration de JGAP :**

  Nous avons du mal à utiliser le framework, malgré les exemples. En effet, certaines fonctionnalités n'étaient pas documentées de manière assez explicite. On était donc obligé d'aller dans le code source du framework pour savoir ce qu'il se passe.

- **Changement dans l'axe d'étude au cours du développement :**

  Au cours du développement et de la rencontre informelle de mi-semestre, nous nous sommes rendus compte que la solution que nous avions proposé au premier semestre n'était pas adaptée.

Malgré les problèmes rencontrés, nous avons réussi à développer une solution opérationnelle. De ce fait nous allons vous présenter les résultats que nous avons obtenu.

4. Résultats et analyses

Pour chaque test, on commence sur une population de première génération (générée aléatoirement).

Sur le cas d'un robot statique (sans déplacement) :

- robot adversaire caractérisé par un déplacement linéaire (match en 1 round) :
  - AGrobot : 57 points (24 % du total) avec 0 points bonus
  - Walls : 181 points (76 % du total)
- même caractéristique mais sur 50 rounds :
  - AGrobot : 8108 points (57%) avec 690 points bonus
  - Walls : 6004 points (43%)
- même caractéristique sur 100 rounds :
  - AGrobot : 16585 points (62%) avec 1600 points bonus
  - Walls : 9992 points (38%)
- pour 200 rounds :
  - AGrobot : 33417 points (63%) avec 3221 points bonus
  - Walls : 20046 points (37%)

A partir de 180 génération, la population commence à stagner.

Le résultat d'un match dépend beaucoup de la position de départ dans laquelle se trouve notre robot.

- robot adverse possède un déplacement aléatoire (1 round) :
  - AGrobot : 157 points (82%) avec 18 points bonus
  - Crazy : 33 points (18%)
- même configuration sur 50 rounds :
  - AGrobot : 6133 points (73%) avec 638 points bonus
  - Crazy : 2279 points (27%)
- même configuration sur 100 rounds :
  - AGrobot : 13721 points (82%) avec 1491 points bonus
  - Crazy : 3111 points (18%)

- même configuration pour 200 rounds :
  - AGrobot : 26604 points (78%) avec 2818 points bonus
  - Crazy : 7442 points (22%)

Tous les résultats dépendent de la valeur du fitness de départ, lors de la génération de la population. De plus, vu que la détermination de l'angle dans lequel doit se positionner la tourelle se calcule de façon linéaire, il est difficile de comparer de manière efficace les résultats des deux cas précédents. Cependant, on voit que la différence de score est peu notable entre l'adversaire Crazy (déplacement aléatoire) et Walls (déplacement linéaire).

IV. Conclusion

D'une manière globale, le projet nous a permis de parfaire nos compétences en algorithmique d'apprentissage.

Pour l'algorithme génétique, les difficultés rencontrées sont surtout dues à des problèmes liés au Framework JGAP et à l'API robocode. Par rapport aux résultats obtenus, il serait préférable d' avoir un moyen de comparaison entre les paliers des différentes solutions. Les résultats actuels ne permettent pas de vérifier si l'algorithme génétique est adapté à ce cas d'utilisation.

Dans le cas du réseau de neurones, par rapport aux résultats obtenus lors des phases d'entrainement, on comprend que cet algorithme n'est pas adapté à cette situation. Car il n'arrive pas à généralisé le problème de façon à ce que les données sortantes soit pertinantes. C'était un challenge d'essayer de mettre en place un réseau de neurones pour l'apprentissage du mouvement, car personne dans la communauté n'avait essayé. Il aurait été plus judicieux d'utiliser le réseau de neurones pour le tir.

Remerciements

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

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

V. 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