

Création d'un robot de combat basé sur un algorithme génétique et un réseau de neurones

RAPPORT FINAL

Alexis BUSSENEAU, Guillaume VAILLAND,
Romain BOIZUMAULT, Vianney MATHIEU
Luc GEFFRAULT

Encadrant : Christian RAYMOND

Résumé

Travail réalisé dans le cadre du cursus ingénieur informatique de l'INSA de Rennes. Notre projet est basé sur l'application Robocode, qui permet aux utilisateurs de coder un robot qui combat d'autres robots. Le but de ce projet est de coder un robot intelligent capable de prendre ses propres décisions en jeu sans se référer à des règles codées manuellement. Pour se faire nous utiliserons un perceptron multi-couches appris lui-même par un algorithme génétique.

1 Notre Objectif

Sur le logiciel Robocode le principe est de coder des règles qui dirigent le robot à la main. Notre objectif est de coder un robot... Mais pas n'importe lequel : Un Robot intelligent. C'est à dire, un robot qui est capable de s'adapter et de s'améliorer. En d'autre termes, c'est un robot capable de créer de lui-même ses propres règles qui influenceront ses décisions au moment du combat.

Pour pouvoir créer un tel robot, nous utiliserons un classifieur spécifique : Le perceptron, un réseau neuronal inspiré du cerveau humain. C'est ce classifieur qui permettra au robot de prendre automatiquement et (normalement) intelligemment ses décisions. Cependant, un classifieur a besoin d'être paramétré correctement pour qu'il puisse être efficace. C'est également le cas du perceptron. Ainsi, nous utiliserons un algorithme génétique afin de le paramétrier. Un algorithme génétique, comme expliqué plus bas, se base sur la théorie de l'évolution de Darwin, d'où notre nom de projet "Robot de Combat Darwinien".

Notre objectif est d'obtenir à la fin du projet un robot capable :

- de s'améliorer par rapport à son comportement de départ,
- de prendre des décisions concernant les actions (tourner, avancer, tirer),
- de s'adapter au terrain et aux ennemis,
- de vaincre un adversaire qui était, avant le traitement, plus fort.

2 Nos Outils

2.1 Robocode

L’application Java au cœur de notre projet. C’est un jeu vidéo éducatif à programmation libre développé par IBM en java. Ce jeu met en scène des tanks virtuels en 2D qui combattent sur un petit terrain. Des tanks de bases sont disponibles mais peu performants, on peut donc créer son propre robot en java avec ses propres actions. Robocode est donc une compétition virtuelle entre plusieurs robots mais aussi une véritable compétition de code entre les joueurs. Vous trouverez plus d’informations sur le site [1].



Déroulement d’une partie - "Build the best destroy the rest" -

2.2 Langage et Formats

Java

Le langage était imposé par le sujet. En effet, Robocode étant codé en java, nous devions réaliser notre travail en java également pour pouvoir le charger dans Robocode.

XML et SSVM

Le projet nous a permis de nous familiariser avec deux types de formats à savoir XML et SSVM. Nous les avons utilisés pour représenter nos différents individus et autres fichiers par soucis de portabilité.

2.3 Environnement de développement

Git

Git nous a été présenté en cours en début d’année. Afin de faire évoluer de manière rapide et coordonnée notre code, nous avons créé un répertoire sur GitHub et utilisé Git à chaque nouvelles mise à jour.

Eclipse et IntelliJ

Le choix d’Eclipse nous a d’abord paru évident puisque nous l’utilisions à chaque TP de Java. Pour simplifier l’utilisation de Git à travers Eclipse nous avons même inclus un plugin nommé EGit. Cependant, la gestion du projet et des conflits Git étant difficiles, nous avons abandonné Eclipse pour IntelliJ, un IDE présenté en TP que nous avons jugé plus complet et plus simple d’utilisation que ce soit pour Git ou pour la création de la JavaDoc.

API de Robocode

Afin d'utiliser les méthodes mise à disposition par IBM à travers Robocode nous avons parcouru de nombreuse fois l'API [2].

3 Notre Travail

3.1 Déroulement d'une partie

Lors du lancement d'une partie de robocode : à chaque tour, notre robot, récupère les valeurs des paramètres de jeux que nous avons trouvé pertinents et que nous appellerons paramètres d'entrée. Nous chargeons ces valeurs dans notre perceptron, dans les neurones d'entrée. Ensuite, le classifieur (le perceptron), que nous avons obtenu grâce à un algorithme génétique, effectue des calculs sur chacun des paramètres d'entrée que nous lui avons demandé de gérer afin d'obtenir différentes valeurs de sortie. Et c'est alors qu'arrive la prise de décision de notre robot. Chaque valeur de sortie est comprise entre -1 et 1 et code une action binaire (Tirer ou non, Tourner à gauche ou non, etc ...). C'est en fonction de cette valeur que notre robot va prendre une décision.

Nous allons maintenant présenter les différents points clefs à savoir : Les données d'entrée, le perceptron et finalement l'algorithme génétique (qui nous permet de paramétrier au mieux le perceptron).

3.2 Données d'entrée

Rôle clé

Les données d'entrée sont les valeurs envoyées au perceptron à chaque prise de décision. Les données d'entrée doivent donc être pertinentes pour que le perceptron puisse prendre de bonnes décisions. Ici, de nombreuses données d'environnements sont accessibles à chaque fois que notre robot scanne un ennemi avec son radar. Certaines de ces données nous sont apparues, de manière évidente, plus ou moins pertinentes. Par exemple, la distance de notre robot par rapport aux ennemis est primordiale mais pas la couleur des robots adverses ni même la nôtre. Il faut également un nombre suffisant de données d'entrée pour que le perceptron puisse être plus précis, puisse manipuler plus de données et établir ses propres règles en fonction d'elles.

Collection

A chaque fois que notre robot scanne un robot adverse à l'aide de son radar il va envoyer les données d'entrée qui peuvent être collectées, et attendre la fin du traitement du perceptron afin de prendre des décisions en fonction des valeurs de sortie de celui-ci. Voici la liste des données que nous pouvions collecter et qui nous ont parues importantes et pertinentes.

- *L'angle de direction de notre radar* relatif à l'angle de direction du robot adverse scanné.
- *La distance* entre notre robot et le robot adverse scanné
- *L'énergie* de notre robot
- *Notre vitesse* ainsi que celle de notre adversaire
- *L'angle de direction* de notre robot ainsi que celui de notre adversaire
- *L'angle de direction de notre radar*
- *L'angle de direction de notre tourelle*

- Nos coordonnées (x, y) dans le plan du jeu
- La distance en x et en y par rapport au robot adversaire scanné

Ainsi, à chaque scan, notre robot intelligent crée un objet "*InputData*" qui contient toutes les données d'entrée en tant que paramètre. Cet objet est chargé dans le perceptron qui va ensuite rendre après traitement un objet "*OutputData*" comportant toute les données de sorties.

Validation par apprentissage supervisé

La difficulté était de se persuader de la pertinence d'une donnée. En effet, même si l'utilisation de la distance paraissait évidente, ce n'était pas le cas de la plupart des autres données récupérables. Par exemple, l'énergie de notre robot ne semblait pas si importante à première vue.

Afin de valider ou non une donnée nous avons donc réalisé une première implémentation du perceptron grâce à un apprentissage supervisé. Cela nous a également permis d'obtenir un premier résultat visuel sur l'amélioration de notre robot.

L'apprentissage supervisé consiste à calculer les paramètres du perceptron grâce à des exemples de prise de décision. La collection d'exemples était seulement possible pour l'action tirer puisqu'il était simple de vérifier si oui ou non notre robot avait bien fait de tirer. (Vérification plus difficile pour un déplacement...).

Nous avons donc créé un robot nommé *AcquisitionBot* qui a pour but de collecter des exemples de données d'entrée lorsqu'il touche son adversaire (C'est à dire lorsque tirer était une bonne décision). Nous avons pu valider la pertinence de nos données d'entrée et en rajouter quelques unes tout au long de l'année grâce à ce type d'apprentissage.

3.3 Perceptron Multicouche

Une fois les données d'entrée récupérées, nous les passons donc dans un perceptron multicouche afin que ce dernier puisse rendre la décision que doit prendre le robot. Regardons donc maintenant de plus près comment se déroule la prise de décision.

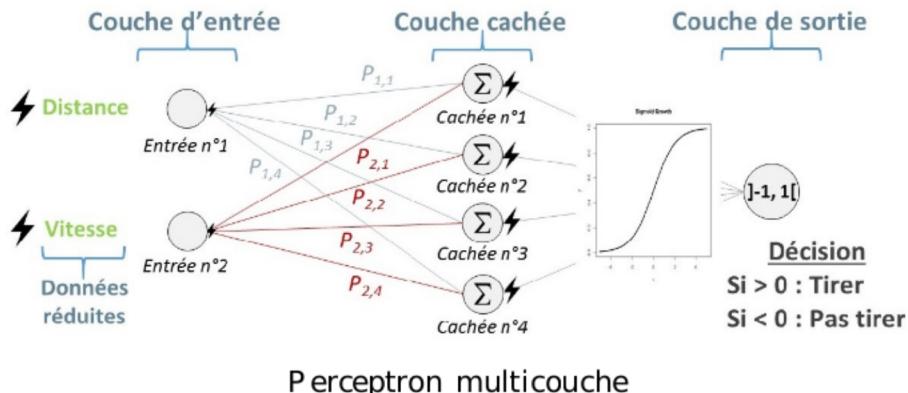
Definition

Le perceptron multicouche est un classifieur de type réseau neuronal formel organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie uniquement. Ce modèle est inspiré des réseaux de neurones humains. Ici, il est composé de uniquement de 3 couches différentes :

- Une couche d'entrée correspondant aux paramètres d'entrée.
- Une couches de neurones cachés.
- Une couche de neurones de sortie correspondant aux paramètres du robot à gérer.

Tous les neurones d'entrée sont connectés aux neurones de la couche cachée, de la même manière tous les neurones de la couche cachée sont connectés aux neurones de la couche de sortie. Chaque connexion est caractérisée par un coefficient de pondération, ce sont les poids du perceptron.

Pour plus d'informations théoriques sur ce classifieur vous pouvez consulter le wikipedia dédié [3]



Fonctionnement

Le perceptron fonctionne de la manière suivante :

- Chaque donnée d'entrée du système implémente une valeur pour chaque neurone de la couche d'entrée.
- Chaque neurone de la couche cachée prend pour valeur la somme des produits des valeurs des neurones d'entrée multipliée par la valeur des poids attribués à chaque connection.
- Chaque neurone de la couche de sortie est le résultat de la multiplication des valeurs des neurones de la couche cachée, auxquelles on a appliqué une fonction sigmoïde, par les poids des liens entre neurones cachés et neurones de sortie.
- Le perceptron prendra sa décision en fonction de la valeur du neurone de sortie.

Implémentation

Nous avons choisi d'implémenter notre type de perceptron suivant 4 classes :

- Une classe définissant les données d'entrée (`inputData`) et dont nous avons choisi d'implémenter la valeur entre -1 et 1.
- Une classe définissant les données des neurones de sortie (`outputData`). Ces neurones prennent la décision qui leur est implémentée si la valeur du neurone de sortie est supérieure à 0 ; sinon il ne fait rien.
- Nous avons donc codé 8 neurones de sortie associés aux actions : Tirer, Avancer, Tourner à gauche (resp. à droite), Tourner la tourelle du robot à gauche (resp. à droite), Tourner le radar à gauche (resp. à droite).
- Une classe implémentant le perceptron lui-même. Nous avons choisi de représenter un perceptron comme l'association de deux matrices, l'une représentant les poids des liens des couches entrée-cachée et l'autre représentant les poids des liens des couches cachée-sortie.

Ainsi, nous avons codé une fonction qui prend comme entrée les valeurs des données d'entrée choisies. Elle les transforme en un vecteur et effectue les applications numériques liées au passage dans le perceptron. On obtient donc en sortie une matrice dont on prendra autant de premiers coefficients qu'il y a de neurones de sortie.

- Une classe définissant les matrices. En effet, nous avons décidé de représenter les poids entre différents neurones sous forme matricielle. Cette implémentation était nécessaire car le logiciel robocode possède une sécurité qui prohibe l'utilisation de librairie externe.

Obtenir un perceptron efficace

Nous avons vu que le perceptron prenait une décision en fonction des données rendues par le perceptron. Ces valeurs de sortie ne dépendent que des données envoyées en entrée et des poids du perceptron. Comme nous désirons obtenir le meilleur robot, nous allons chercher à trouver les poids les plus adéquats pour "paramétrier" notre perceptron. Nous cherchons donc, respectivement à notre implémentation, à trouver les matrices de poids les plus efficaces pour nos décisions. Pour ce faire, nous allons utiliser un algorithme génétique.

3.4 Algorithme Génétique

Comme dit précédemment, l'algorithme génétique va nous permettre de fixer les bons poids du perceptron, c'est à dire de paramétriser le perceptron afin qu'il soit efficace et rende de bonnes valeurs de sorties.

Fonctionnement et implémentation

Cet algorithme se base sur la théorie de l'évolution de Darwin comme l'indique le titre de notre projet. Son principe est simple : Croiser des robots intelligents 2 à 2 pour leur créer deux "descendants" (dans notre cas un descendant étant un mélange de poids des matrices de poids représentants chacun des deux perceptrons "parents").

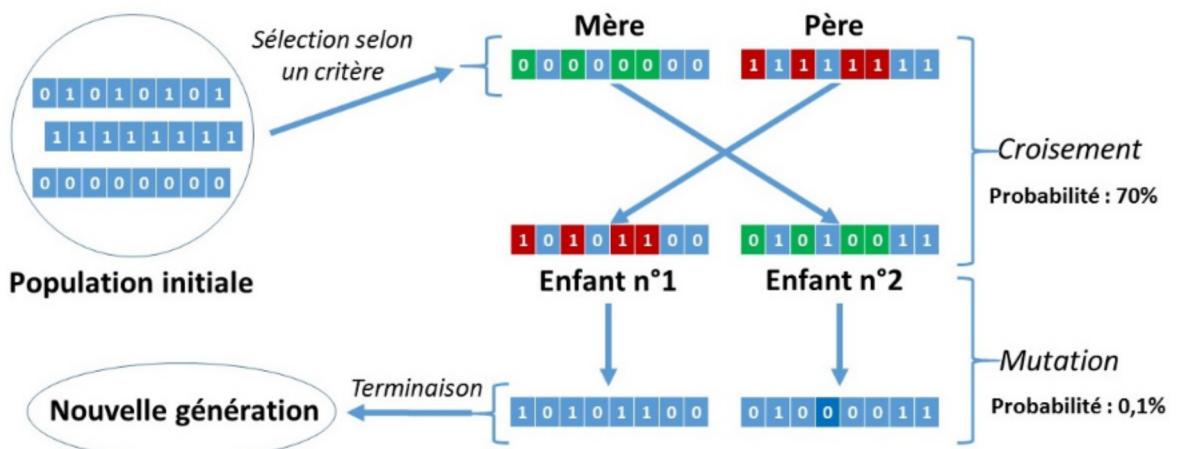


Schéma de l'Algorithm Génétique

Pour ce faire, nous partons d'une population initiale de n individus (n matrices de poids représentants les perceptrons). On garde le meilleur de la population initiale (n). Puis nous prenons au hasard q individus parmi les n de départ et nous les faisons s'affronter les uns contre les autres. On garde le meilleur de ce tournoi entre q individus. On répète l'étape du tournoi de façon à obtenir $\frac{n}{2}$ individus sélectionnés par tournoi. On effectue alors croisements et mutations des individus deux à deux.

Dans la sélection par tournoi nous avons choisi de surcharger la fonction *compareTo* de Robocode et de retourner l'individu ayant le plus grand pourcentage de victoire. Et si jamais 2 individus arrivent à égalité, on regarde lequel des deux a fait le plus de dégâts directs.

Pour plus d'informations théoriques sur cet algorithme vous pouvez consulter le wikipédia dédié [4].

Lancement de l'algorithme génétique

Une fois notre algorithme génétique implémenté, il ne reste plus qu'à le lancer, afin d'espérer paramétrier au mieux notre perceptron. Comme nous l'avons déjà expliqué, cet algorithme va donc prendre en entrée 2 matrices de poids : une pour les poids entre les neurones d'entrée et les neurones de la couche cachée, une autre pour les poids entre ces derniers neurones et les neurones de sortie. L'algorithme effectue donc le "croisement génétique" entre ces matrices et nous obtenons à la fin un perceptron correctement paramétré, avec les meilleurs poids possibles. Ces derniers sont en fait un mixe des meilleurs poids des différents perceptrons brassés par l'algorithme.

4 Notre expérience

4.1 Répartition du travail

Tout au long du projet nous nous retrouvions (au mieux) toutes les semaines pendant 2h afin de faire le point sur l'avancement du projet, se mettre à jour et partager nos connaissances sur le sujet ou sur les outils.

Premier Semestre

- Dans un premier temps nous nous sommes tous concentrés sur la compréhension du fonctionnement du perceptron, de l'apprentissage supervisé, ainsi que du sujet dans sa globalité. Une étape cruciale et complexe qui s'étala sur tout le premier semestre en parallèle du reste.
- Nous nous sommes également familiarisés avec l'application Robocode notamment grâce à l'API [2]
- Ensuite, nous nous sommes répartis le travail en deux groupes. D'un côté, Alexis et Romain se sont occupés de coder un perceptron. De l'autre, Luc s'est occupé de rechercher les données d'entrée fournies par Robocode et en particulier celles exploitables pour notre apprentissage. Guillaume a travaillé sur les deux plans tout en réalisant la liaison entre le code et l'avancement théorique de Luc.
- Enfin nous avons implémenté *AcquisitionBot*, qui nous a permis de réaliser l'algorithme supervisé.
- Vers la fin du semestre, nous nous sommes tous attelés à la préparation de la soutenance.

Second Semestre

- Après avoir apporté leur aide sur le développement du code, Luc et Guillaume ont retravaillés les données d'entrée afin d'améliorer le rendement du robot et d'essayer de palier au problème de centrage des données.
- Alexis et Vianney se sont concentrés sur la réalisation et l'implémentation de l'algorithme génétique.
- Romain lui s'est occupé de gérer la lecture de fichier entre ceux reçus de la part de Robocode et ceux envoyés à Robocode.
- Une fois le travail de l'un terminé, il apportait son aide à une autre équipe ou commençait la rédaction du rapport. Ainsi, Luc a repris le code du perceptron

- afin de l'adapter à l'apprentissage non supervisé et Guillaume commencé la rédaction de la JavaDoc.
- Pour finir, Alexis, Guillaume, Romain et Luc ont terminé la rédaction des 3 rapports du projet et préparé la soutenance finale.

4.2 Ressenti

Bien évidemment, ce projet nous a permis de travailler en groupe au sein d'un même projet. C'est avant tout une expérience qui sera réitérée à l'avenir pour la majorité d'entre nous au sein du monde professionnel. Ainsi, nous avons appris à travailler avec les autres, mais aussi à prendre des responsabilités et àachever une tâche attribuée de manière autonome. Il nous est arrivé de rencontrer quelques tensions notamment durant les périodes clés qui précédaient les soutenances mais nous avons tous surmonté de manière adulte et professionnelle ces petites querelles et ce projet annuel reste une très bonne expérience. Nous avons pu tester nos abilités et développer nos capacités et connaissances.

5 Conclusion

D'un point de vue technique nous avons réussi à construire un algorithme génétique avec sélection par tournoi fonctionnel. Notre robot final, quant à lui, arrive à prendre des décisions en fonction de son environnement. Il décide de tourner, d'avancer ou de tirer de manière autonome. Cependant, bien qu'ayant implémenter toutes les parties théoriques du projet, la réalisation pratique laisse encore à désirer. En effet, actuellement, notre perceptron donne des valeur de sorties très proches, peu représentatives d'une véritable décision binaire. Cela peut être dû à plusieurs choses au niveau de chaque étape du déroulement global. Ainsi, de nombreuses améliorations sont encore possibles au niveau des données d'entrée (rajout/suppression de données, centrage et réduction), de l'algorithme génétique (ajuster les probabilités) ou encore de l'implémentation des décisions (changer la prise de décision, changer les seuils de décision).

Sur le plan humain, nous sommes tous satisfait d'avoir pu participer à un tel projet même si les résultats techniques sont assez mitigés. En effet, nous avons pu découvrir en profondeur des théories complexes comme celle de l'algorithme génétique ou du perceptron multicouche. Pour finir, plus axé recherche que production, ce projet nous a permis d'acquérir et de consolider plusieurs connaissances. Une véritable expérience à refaire.

Références

- [1] IBM. Site officiel de robocode. <http://robocode.sourceforge.net/>.
- [2] IBM. *Robocode 1.9.2.0 API*, 2014.
- [3] Wikipedia. Perceptron multicouche. https://fr.wikipedia.org/wiki/Perceptron_multicouche.
- [4] Wikipedia. Algorithme génétique.
https://fr.wikipedia.org/wiki/Algorithme_g%C3%A9n%C3%A9tique.

