Création d'un robot de combat darwinien

Simon BEAULIEU, Edgar SERRANO,
Raphaël ESTEVENY, Martin GOUBET

Encadrant : Christian RAYMOND

May 22, 2017

Résumé

Travail réalisé dans le cadre du cursus ingénieur informatique de l'INSA de Rennes. Le projet est réalisé à l'aide de Robocode, application permettant aux utilisateurs de coder un robot combattant d'autres robots. L'objectif de ce projet est d'obtenir un robot capable de prendre des décisions par lui-même au travers d'un perceptron, appris par un algorithme génétique.

1 Objectifs

Le logiciel Robocode permet d'implémenter des règles qui définissent le comportement de notre robot. Notre objectif principal est donc d'obtenir un robot intelligent. C'est-à-dire un robot capable de s'adapter et de s'améliorer. Afin de créer ce robot, nous utiliserons un perceptron ou réseau de neurones. Cela permettra à notre robot de prendre automatiquement et intelligemment ses décisions durant le jeu. Le perceptron nécessite cependant un paramétrage spécifique afin d'être efficace. Pour ce faire nous utiliserons un algorithme génétique pour le paramétrer, algorithme basé sur la théorie de l'évolution de Darwin, expliquant le nom du projet "Robot de Combat Darwinien".

À la fin du projet notre robot devrait être en mesure de :

- s'améliorer par rapport au comportement initial.
- s'adapter au terrain et aux ennemis.
- vaincre un adversaire de plus en plus fort.

2 Outils

2.1 Robocode

Il s'agit d'un jeu vidéo éducatif à programmation libre en java développé par IBM. Ce jeu met en scène des tanks virtuels en 2D qui combattent sur un champs de bataille. Il est possible d'utiliser des tanks disponibles par défaut, bien que peu performants. Nous pouvons créer notre propre robot en java avec ses propres actions. Vous trouverez plus d'informations sur le site dédié de Robocode. [1]

2.2 Langage et Format

2.2.1 Java

Le langage était imposé par le sujet. En effet, Robocode est codé en Java et afin d'ajouter notre robot à Robocode nous devions également le réaliser en Java.

2.2.2 XML

Nos différents individus générés par l'algorithme génétique sont représentés en XML.

2.3 Environnement de développement

2.3.1 Git

Git nous a été présenté en début d'année. Nous avons donc choisi de l'utiliser afin d'avancer notre projet de manière simple et de coordonner notre code, nous avons créé un dépôt distant sur GitHub pour mettre à jour notre code.

2.3.2 IntelliJ

On nous a présenté l'IDE IntelliJ, qui nous avait semblé être approprié pour notre projet. De plus le groupe de l'année dernière nous a fortement conseillé de l'utiliser car il est plus simple d'utilisation pour lier notre code Java à Robocode. C'est pourquoi nous avons opté pour IntelliJ.

2.3.3 API de Robocode

Afin de comprendre toutes les méthodes que nous pourrions utiliser dans Robocode, nous avons parcouru la documentation mise à disposition par IBM.

3 Notre Travail

3.1 Déroulement d'une partie

Une partie de Robocode se décompose en une succession de tours pendant lesquels notre robot récupère les données de son environnement qui nous ont semblées pertinentes afin de prendre des décisions.
Nous allons maintenant présenter les différents points clés, à savoir :

- les données d'entrée.

- les données de sortie
- le perceptron
- l'algorithme génétique

3.2 Les données d'entrée

3.2.1 Rôle

Ces données sont récupérées par notre robot à chaque tour de jeu. Elles sont prises en compte par notre perceptron, dans les neurones d'entrée. Le perceptron effectue ensuite une série de calculs sur ces données afin d'obtenir différentes valeurs de sortie. Ces valeurs permettront à notre robot de prendre des décisions.
Ces données sont donc très importantes pour notre robot et se doivent d'être pertinentes afin que le perceptron puisse prendre les bonnes décisions au bon moment.
Grâce à Robocode nous avons de nombreuses données d'environnement lorsque nous scannons un adversaire avec notre radar. Certaines sont assez évidentes comme par exemple la distance entre notre robot et l'adversaire scanné, extrêmement importante. Cependant la couleur des robots n'est pas nécessaire pour prendre de bonnes décisions. De plus, afin d'avoir un perceptron suffisamment précis, il faut disposer d'un nombre suffisant de données d'entrée pour qu'il puisse manipuler plus de données et établir ses propres règles déduites.

3.2.2 Collection

A chaque fois que notre robot scanne un adversaire, il va envoyer les données collectées au perceptron pour prendre ses décisions en fonction des valeurs de sorties. Voici la liste des données que nous pouvons collecter et que nous avons trouvé pertinentes pour notre robot:

- *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 vélocité* 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 y par rapport au robot adversaire scanné*
- *La distance en x et y par rapport aux différents murs* vers lequel se dirige le robot

Ainsi, à chaque scan, notre robot crée un objet "InputData" qui contient toutes ces données. Nous chargeons ensuite cet objet dans le perceptron qui renverra, après traitement, un objet "OutputData".

3.3 Données de sortie

Les données de sortie sont les données que l'on récupère une fois que nos données d'entrée ont été traitées par le perceptron. Ces données représentent donc les actions que notre robot peut choisir d'effectuer. De ce fait, ces données sont également très importantes. Chaque valeur de sortie est comprise entre -1 et 1 et permet de réaliser une action (tirer, tourner...). De plus ces valeurs ont une influence sur la façon dont se déroule une action, comme par exemple, tourner à droite d'un certain angle. Ainsi, à chaque traitement du perceptron, notre robot peut choisir d'effectuer les décisions suivantes :

- *Tirer*
- *Tourner à droite* d'un certain angle
- *Tourner à gauche* d'un certain angle
- *Tourner le radar et la tourelle à droite* d'un certain angle
- *Tourner le radar et la tourelle à gauche* d'un certain angle
- *Avancer*

Contrairement aux années précédentes, nous avons choisi de lier la décision de la rotation du radar et celle de la tourelle pour éviter que le robot ne tire alors que la tourelle n'était pas en bonne position. Ainsi notre robot traitera l'objet « Output-Data » qui lui est renvoyé afin de prendre ses décisions.

3.4 Perceptron

Comme dit précédemment, les données d'entrée sont récupérées et passées dans un perceptron multicouche afin que ce dernier puisse nous renvoyer les données de sortie et que notre robot puisse ainsi prendre ses décisions. Nous allons donc regarder comment fonctionne notre perceptron.

3.4.1 Définition

Un perceptron multicouche est un classifieur dit neuronal formel organisé en plusieurs couches au sein desquelles une information circule des couches d'entrée vers les couches de sortie. Ce modèle s'inspire du fonctionnement de nos neurones. Dans notre cas nous utilisons trois couches différentes :

- Une couche d'entrée qui récupère nos données d'entrée
- Une couche de neurones cachés
- Une couche de sortie qui nous permet de prendre nos décisions

Dans un perceptron, tous les neurones de chaque couche sont connectés à tous les neurones de la couche suivante. Ainsi, tous les neurones d'entrée sont connectés aux neurones de la couche cachée. De même, les neurones de la couche cachée sont connectés aux neurones de sortie. L'important dans un perceptron est que chacune de ces connexions entre neurones est caractérisée par un coefficient de pondération, qui représentent les poids du perceptron. Pour plus d'informations théorique sur ce classifieur, vous pouvez consulter le wikipédia dédié. [2]

3.4.2 Fonctionnement

Le perceptron fonctionne de la manière suivante :
Chaque donnée d'entrée est représentée par un neurone dans 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 connexion.
Chaque neurone de la couche de sortie correspond à une action du robot. Il est le résultat de la multiplication des valeurs des neurones de la couche cachée, auxquelles nous avons appliqué une fonction Relu, par les poids des liens entres neurones cachés et neurones de sortie.
Ensuite, notre robot prend sa décision en fonction de la valeur des neurones de sortie.

3.4.3 Implémentation

Le perceptron est implémenté en quatre classes java : Une classe définissant les données d'entrée (InputData) et dont la valeur est comprise 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. Dans le cas où l'on souhaite tourner d'un certain angle, plus la valeur du neurone est élevée, plus l'angle de rotation sera élevé.
Une classe implémentant le perceptron. Un perceptron est représenté comme l'association de deux matrices, l'une représentant les poids des liens des couches entrée-cachée et l'autre les poids des liens des couches cachée-sortie.
Ainsi nous disposons d'une fonction qui prend les valeurs des données d'entrée choisies et les transforme en un vecteur et effectue les différentes applications numériques. On obtient donc une matrice dont on prendra autant de premiers coefficients qu'il y a de neurones de sortie.
Une classe définissant les matrices. En effet, les poids des différents neurones sont représentés sous forme matricielles.

3.4.4 Obtenir un perceptron efficace

Nous avons vu que le robot prenait une décision en fonction des données rendues par le perceptron. Et ces valeurs de sortie ne dépendent que des données envoyées en entrée et des poids du perceptron. Afin d'obtenir un bon robot, nous devons donc paramétrer des poids adéquats sur chaque connection de notre perceptron. Pour ce faire, nous allons donc chercher à obtenir les matrices de poids grâce à un algorithme.
Une autre chose importante afin d'avoir un perceptron efficace est d'avoir des données pertinentes en entrée, de même pour les données de sortie.

3.5 Algorithme génétique

Comme dit précédemment, nous avons besoin d'obtenir les matrices de poids et nous allons donc nous servir d'un algorithme génétique afin de paramétrer le perceptron de manière efficace.

3.5.1 Fonctionnement et implémentation

Cet algorithme se base sur la théorie de l'évolution de Darwin. Le principe fondamental est assez simple : on croise les robots d'une même génération d'individus

2 à 2 afin de créer des descendants, c'est-à-dire un mélange des matrices de poids des perceptron parents.

Pour ce faire, nous partons d'une population initiale de n individus (n matrices de poids représentant les perceptrons). Pour garder uniquement les meilleurs individus de la population initiale, nous prenons au hasard q individus que nous faisons s'affronter contre un même robot témoin lors d'une bataille de robocode de 10 rounds. Nous gardons les meilleurs de ce tournoi. On répète cette étape jusqu'à obtenir n/2 individus sélectionnés. Nous effectuons alors croisements et mutations des individus 2 à 2 pour obtenir la population suivante.
Le groupe de l'année dernière a choisi d'utiliser la fonction compareTo de robocode afin de retourner le robot avec le plus grand pourcentage de victoire. Si jamais 2 individus arrivent à égalité, est gardé celui qui aura fait le plus de dégats directs. Cependant, cette méthode n'était pas assez précise pour bien distinguer les forces des différents robots. C'est pourquoi nous avons choisi de pondérer le score des robots de manière arbitraire, c'est-à-dire que nous avons analysé les composantes du score de robocode et nous avons choisi de ne garder que les dégats du canon (BulletDammage), les dégats par collision (RamDammage) ainsi que le nombre de matchs remportés (Victory). Nous avons également pondéré ces valeurs en donnant plus d'importance aux BulletDammage et à Victory qu'aux RamDammage.
Pour plus d'informations concernant l'algorithme génétique, vous pouvez consulter le wikipédia dédié. [3]

3.5.2 Lancement de l'algorithme génétique

Une fois notre algorithme génétique pondéré et implémenté, il nous reste à le lancer, afin de paramétrer au mieux notre perceptron. Comme nous l'avons dit, cet algorithme va prendre en entrée deux matrices de poids : une pour les poids d'entrée et l'autre pour les poids de sortie. Notre algorithme génétique effectue ensuite le croisement de ces matrices et nous obtenons à la fin un perceptron correctement paramétré, avec un ensemble de poids correspondant aux meilleurs poids des différents perceptrons brassés par l'algorithme.

4 Expérimentation

4.1 Tests

Durant ce projet, nous avons dû effectuer de nombreux lancements de l'algorithme génétique afin de tester nos améliorations. Pour ce faire nous avions deux méthodes de test. Si nous voulions un résultat rapidement nous lancions l'algorithme avec 5 générations de 10 individus. Cependant afin d'avoir de meilleurs résultats nous devions lancer l'algorithme avec 50 générations de 100 individus. De plus, au début de l'année nous utilisions un robot "inactif", c'est-à-dire un robot qui ne tire pas et qui ne bouge pas, pour nos tests. Cependant, au fur et à mesure de l'avancée du projet nous avons remarqué que nos robots battaient facilement cet adversaire. Nous sommes donc passé à un robot "aléatoire", un robot dont tout le comportement est aléatoire.

4.2 Résultats

Grâce aux nombreux tests, nous avons pu obtenir des résultats intéressants. En effet, nous avons pu remarquer sur les lancements à 50 générations que le score des

robots augmentait au fur et à mesure des générations. Cependant cette augmentation reste assez petites et pourrait être améliorée. Nous avons également obtenu différents comportements de robot. Par exemple, un robot se déplaçant sur le champ de bataille en tirant sur son adversaire, un comportement de base. Nous avons également obtenus deux autres comportements intéressants, un robot « kamikaze » qui repère son adversaire et lui fonce dedans afin d'infliger des dégâts de collision, et un robot « lâche » qui se contentait de fuir son adversaire.

4.3 Conclusion sur notre robot

Par rapport aux objectifs de début d'année, nous avons réussi à obtenir un robot meilleur que le robot initial. En effet, comme dit précédemment, nous avons commencé par essayer de réaliser un robot qui battait un robot "inactif" et nous réussissons maintenant à battre un robot au comportement aléatoire et certains autres robots aux comportements plus scriptés. De plus, notre robot adapte son comportement aux adversaires. En effet, quand notre meilleur robot affronte le robot "inactif", il adopte une stratégie où il rentre en collision avec l'ennemi tout en lui tirant dessus ce qui le fait gagner automatiquement. Par contre, quand il affronte le robot au comportement aléatoire, il se déplace sur le champ de bataille et ne tire que quand il détecte l'ennemi. De même, nous avons commencé à implémenter l'adaptabilité de notre robot au terrain en essayant de lui faire éviter les murs du champs de bataille.

5 Notre expérience

5.1 Répartition du travail

Durant toute l'année, nous nous sommes retrouvés en groupe une fois toutes les deux semaines, afin de mettre en commun nos avancées personnelles et les différentes connaissances que nous avions acquises sur le sujet ou sur les outils.

5.1.1 Premier Semestre

Pour ce faire nous avons étudié le code du projet durant tout le semestre afin de se l'approprier en parallèle des autres objectifs. Lors du premier semestre, nos objectifs étaient de comprendre la façon dont le groupe précédent avait codé le perceptron et l'algorithme génétique, lancer l'algorithme génétique et commencer à réfléchir aux différentes améliorations.
Nous avons utilisé Robocode afin de savoir comment utiliser le robot résultant de l'algorithme génétique. Nous avons aussi corrigé plusieurs bugs empêchant l'algorithme de se lancer.
Vers la fin du semestre nous nous sommes focalisés sur la soutenance à faire en anglais.

5.1.2 Second Semestre

Durant le premier semestre nous avions commencé à réfléchir à différentes améliorations du robot. Afin de les réaliser, nous avons réparti le travail entre 2 groupes.
Edgar et Raphaël avaient pour but d'améliorer les données d'entrée, c'est-à-dire de juger de la pertinence de ces données et de regarder s'il ne serait pas mieux d'en enlever certaines, ou alors d'en rajouter d'autres.

Simon et Martin, quant à eux, se sont occupés de finir le débogage de l'algorithme ainsi que de la pondération du score de chaque robot. Ces travaux ont été effectués tout en continuant à nous rencontrer ensemble afin de mettre en commun nos avancées respectives.
Martin a également créé et géré un Github afin que l'on puisse coder en parallèle durant le semestre.

5.2 Conclusion

Ce sujet d'étude pratique nous a permis d'apprendre à mieux travailler en équipe au sein d'un même projet comme nous aurons à le faire en milieu professionnel. Ainsi, nous avons appris à travailler ensemble, à avoir des responsabilités au sein d'une équipe et d'accomplir nos tâches respectives. Nous avons également acquis des compétences qui nous serons utiles dans notre vie professionnelle comme l'utilisation de git ou une expérience supplémentaire de travailler en Java et XML.
Nous avons malheureusement eu quelques soucis en début de 2ème semestre car le projet continuait de bugger sur certains ordinateurs, ce qui a conduit à une baisse de motivation. Cependant après quelques semaines nous avons réussi à débogger le projet pour tout le monde et nous avons pu reprendre le travail.
Malgré quelques tensions aux moments des soutenances, ce projet reste une expérience enrichissante durant laquelle nous avons pu mettre en pratique les différentes connaissances acquises en cours tout au long de l'année.
Au final, nous avons tous été satisfait de participer à un tel projet. En effet cela nous a permis de découvrir et de comprendre différents algorithmes assez complexes comme l'algorithme génétique.

6 Références

[1] IBM. Site officiel de robocode:
`http://robocode.sourceforge.net/`
[2] Wikipédia. Perceptron multicouche:
`https://fr.wikipedia.org/wiki/Perceptron_multicouche`
[3] Wikipédia. Algorithme génétique:
`https://fr.wikipedia.org/wiki/Algorithme_g%C3%A9n%C3%A9tique.`