Darwini

Daniel Haus, Etienne Rebout, Antoine Barroux, Auriane Gilbert

May 2018

Encardrant : Christian RAYMOND

Sommaire

1 Introduction

Ce projet est mené dans le cadre des études pratiques au sein de notre formation à l'INSA de Rennes. Nous utilisons une application nommée Robocode qui nous permet de créer un robot pour le faire se battre contre d'autres robots fournis par l'application. 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.

2 Objectifs du projet

Notre projet a pour but de créer un robot qui apprenne seul à se battre pour qu'il puisse d'adapter aux différents adversaires qui lui feront face. Ce robot est donc doté d'un perceptron ou réseau de neurones qui lui permettra de réfléchir et d'agir selon différentes situations. A la fin du projet nous attendons que notre robot obtienne de meilleurs scores et batte plus des enemis plus forts que lorsque nous avons repris ce projet.

3 Outils utilisés

3.1 Robocode

Robocode est un jeu vidéo à but éducatif créé et distribué par IBM destiné à l'apprentissage du langage Java. Il nous permet de créer notre propre robot et de le faire se battre contre un ou plusieurs robots par défaut fournis par le jeu. Les robots sont représentés sous forme de tanks qui combattent dans un terrain.

3.2 Langage et format

3.2.1 Java

Comme expliqué precédemment, Robocode a été créé dans le but d'apprendre le Java donc notre robot a été implémenté en Java.

3.2.2 XML

Le perceptron de chaque individu est stocké en XML dans le projet.

3.3 Environnement de développement

3.3.1 IntelliJ

L'IDE IntelliJ était utilisé par le groupe précédent et nous avait été présenté en cours, de plus il nous semblait plus simple d'utilisation qu'Eclipse c'est pourquoi nous l'avons choisi.

3.3.2 GitHub

Nous avons créé un répertoire GitHub afin de pouvoir travailler autant chez nous qu'à l'INSA. De plus Git gère le fait que plusieurs personnes travaillent simultanément sur le même fichier ou permet de dupliquer le projet en "branche" lorsqu'une modification importante est en cours mais qu'elle ne fonctionne pas encore ce qui permet à la fois d'avoir un prototype fonctionnel et un prototype en développement. Nous avons choisi GitHub contrairement à GitLab car nous n'avons pas trouvé de projet similaire sur internet et nous serions ravis si notre travail pouvait aider certaines personnes à découvrir Robocode ou les réseaux de neurones.

3.3.3 API de Robocode

La documentation de Robocode fournie par IBM est à la base de ce projet.

3.3.4 Trello

Nous avons utilisé Trello, un outil de gestion de projet gratuit en ligne qui permet de créer des tâches dans trois tableaux : A faire, en cours et fait. Il y a également la possibilité d'assigner des tâches à certaines personnes ou encore d'ajouter une date limite à certaines tâches, et cela permet visuellement de voir l'avancement du projet.

4 Travail effectué

4.1 Le fonctionnement du jeu

Une partie de Robocode commence par la création d'un robot puis il est envoyé dans un combat dans lequel il doit récupérer des informations, et utiliser son perceptron afin de prendre des décisions. Notre robot est constitué d'un radar afin d'apercevoir l'enemi, d'un canon afin de tirer sur son adversaire et de chenilles pour se diriger. Ces trois composants peuvent tourner indépendamment les uns des autres. Nous allons aborder plus en détails comment le robot récupère les informations, le fonctionnement du perceptron, la prise de décision et le fonctionnement de l'algorithme génétique.

4.2 Les données d'entrées

4.2.1 Rôle

Les données récupérées par le robot sont essentielles à la prise de décisions car elles sont les neurones d'entrée du perceptron. Le robot récupère des informations importantes pour le jeu telles la position de l'enemi ou encore le niveau de sa barre de vie et des informations secondaires telles la couleur des adversaires. La difficulté consiste à trier ces informations et ne garder que les plus pertinentes. De plus, il a fallut modifier certaines informations, par exemple Robocode nous fournit les coordonnées de l'adversaire mais nous avons remarqué que notre robot est bien plus performant si nous lui donnons plutôt la distance qui le sépare de son adversaire. Cela vaut également pour la distance entre notre robot et le mur le plus proche afin qu'il évite de cogner dans les murs pendant toute la partie.

4.2.2 Données choisies

Les données que nous avons choisit de garder sont les suivantes :

- ll'angle de direction de l'adversaire
- L'angle de direction de notre radar relatif à l'angle de direction du robot adverse scanné.
- la distance entre notre robot et l'adversaire
- l'énérgie restante à notre robot
- la vitesse de l'adversaire
- notre vitesse
- notre angle de direction
- l'angle de direction de notre radar
- l'angle de direction de notre canon
- la distance entre notre robot et le mur le plus proche

4.3 Les données de sortie

4.3.1 Rôle

Les données de sortie sont les décisions que le robot aura prises à l'aide du perceptron. Pour permettre à notre robot d'apprendre plus facilement, on ne lui apprends que quelques décisions à la fois, par exemple nous avons privilégié le déplacement du robot (ne pas se cogner dans les murs, éviter les balles de l'adversaires, etc) à la décision de tirer car sinon notre robot restait immobile et tirait sur l'adversaire en continu. Sachant que sur Robocode, le robot qui tire sur son adversaire perds de la vie à chaque tir, cette conduite était suicidaire.

4.3.2 Données choisies

Les données de sortie retenues sont :

- tirer sur l'adversaire ou non
- angle de direction
- angle de direction du radar
- angle de direction du canon
- avancer

4.4 Perceptron

Le perceptron est ce qui lie les données d'entrée aux données de sortie.

4.4.1 Définition

Un perceptron multicouche est un classiffieur 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 :

- la couche d'entrée qui récupère nos données d'entrée
- la couche de neurones cachés
- la couche de sortie qui nous permet de prendre des 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.

4.4.2 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.

4.4.3 Implémentation

Le perception est implémenté en 4 classes Java.

- Une classe InputData qui récupère les données fournies par le jeu. Les neurones d'entrées sont donc les valeurs numériques fournies par le jeu.
- Une classe OutputData qui définit les neurones de sortie donc les décisions du robot. Par exemple le neurone qui correspond au tir peut contenir la valeur 1 ce qui signifie que le robot doit tirer ou la valeur 0 pour ne pas tirer.
- Une classe NeuralNetwork 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. Ces poids sont générés de manière aléatoire. 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 matrix car les poids des différents neurones sont présentés sous forme matricielle.

4.5 L'algorithme génétique

Pour obtenir un robot performant, il faut donc que les poids du perceptrons soient adaptés. L'intérêt de l'algorithme génétique est que le robot va trouver lui-même la combinaison idéale des poids. Pour ce faire nous allons utiliser la théorie de l'évolution.

4.5.1 La Fitness

Pour commencer il nous faut savoir évaluer la performance d'un robot lors de son combat. Pour cela nous utilisons une méthode baptisée "fitness" qui, a la fin d'un combat nous génère un résultat qui sera le score du robot. Cette fitness peut être changée en fonction des résultats que l'on veut obtenir, par exemple si l'on souhaite que notre robot évite les murs on peut enlever des points au robot à chaque fois qu'il tape dans un mur, ainsi lorsqu'on gardera le meilleur individu ce sera forcément celui qui a le moins tapé dans un mur. Notre fitness actuelle est la suivante :

```
  weightedScore = 6 * bulletDamage + 6 * survival
+   1 * ramDamage + 300 * (hits / (hits + missed));
```

Elle est donc influencée positivement par les dégâts causés à l'adversaire, par le canon ou par collision, le fait de gagner la partie, et le taux de tirs réussis.

4.5.2 Populations et individus

Nous allons donc créer une population, c'est à dire un ensemble d'individus et les faire combattre. grâce à la fitness, nous allons pouvoir classer ces individus du plus fort au plus faible et conserver les meilleurs individus. Les poids des individus de la nouvelle génération seront calculés en fonction de la moyenne et de l'écart-type des poids des meilleurs individus de l'ancienne génération. Nous incluons également le meilleur individu de la génération précédente dans la nouvelle génération. Cet algorithme nous permet après maintes générations de populations d'obtenir le meilleur robot.

5 Expérimentations et tests

A compléter après les derniers tests

6 Notre expérience

6.1 Premier semestre

Le permier semestre a été principalement consacré à la compréhension du projet et au debogage du code du groupe précédent. Nous avons restructuré le code et redéfinit certaines fonctions ainsi que la fitness et effectué quelques tests.

6.2 Second semestre

Lors du second semestre nous avons changé le fonctionnement de l'algorithme génétique, qui générait initialement la population suivante à partir de deux individus et non pas à partir des moyennes et ecart-types des meilleurs individus. Nous avons notamment ajouté le fait d'inclure le meilleur individu d'une population à l'autre, modifié les fonctions appliquées dans le perceptron, les données d'entrées ou de sorties.

6.3 Répartition du travail

Nous avons organisé des séances de travail collectives et pris des rendez-vous avec notre encadrant régulièrement, notamment lors d'avancées ou de décisions importantes. Le répertoire Git et le tableau de tâches sur Trello nous a permis de travailler chacun de notre côté lorsque nous avions du temps libre.

6.4 Conclusion

Ce projet nous a permis d'améliorer nos compétences de travail en équipe, de comprendre cxe qu'était un réseau de neurones, d'améliorer nos méthodes de gestion de projet via l'outil Trello et de mettre en pratique nos connaissances en Java et XML. il nous a également appris à monter en compétence sur un sujet qui nous était inconnu de manière quasiment autonome car une fois le principe de base expliqué il a fallu déterminer quels algorithmes étaient les plus adaptés ou quelles solutions étaient viables. Nous avons tous été très satisfaits de participer à ce projet car il nous a permis de découvrir un domaine passionnant et d'actualité.