Réseau de Neuroévolution Appliqué au Jeu Vidéo

Thomas Ibanez

6 juillet 2018

printemps 2018
Session de bachelor

INGÉNIERIE DES TECHNOLOGIES DE L'INFORMATION

ORIENTATION – LOGICIELS ET SYSTEMES COMPLEXES

ALGORITHMES GENETIQUES POUR L'INTELLIGENCE ARTIFICIELLE

Descriptif :

Le développement des algorithmes et l'augmentation de la puissance de calcul à faible prix a permis un grand développement de l'intelligence artificielle (IA) dans les dernières années. Une application typique de l'IA est les jeux et certaines victoires récentes contre les meilleurs humains ont démontré le grand potentiel le l'apprentissage automatique dans le domaine (go, poker, dota, ...). Dans ce projet, le but est de se familiariser avec les méthodes d'apprentissage par réseaux de neurones et des algorithmes génétiques. Pour ce faire, l'apprentissage se fera sur un jeu simple (asteroid) dont les entrées et sorties sont très limitées en nombre. Deux méthodes basée sur des algorithmes génétiques seront utilisées. La première consistera à utiliser un perceptron multicouches, dont les poids seront optimisés à l'aide d'un algorithme génétique. La seconde est l'implémentation d'une méthode NEAT (Neuroevolution of augmenting topologies
) qui optimisera non seulement les poids mais également la topologie du réseau de neurones.

Travail demandé :

En fonction du temps à disposition, l'étudiant devra effectuer les tâches suivantes :

- Écriture du jeu asteroid.
- Implémentation d'un perceptron.
- Entraînement à l'aide d'un algorithme génétique du réseau de neurones.
- Analyse des résultats.
- Implémentation de la méthode NEAT.
- Entraînement à l'aide de la méthode NEAT.
- Analyse des résultats et comparaison avec l'apprentissage par algorithme génétique.

Candidat :

M. Ibanez Thomas

Filière d'études : ITI

Professeur(s) responsable(s) :

MALASPINAS ORESTIS

En collaboration avec :
Travail de bachelor soumis à une convention de stage en entreprise : **non**
Travail de bachelor soumis à un contrat de confidentialité : **non**

Table des matières

Table des figures

Remerciements

Acronymes

IA NEAT

1 Introduction

Ces dernières années l'apprentissage automatique (Machine Learning) à pris une place très importante dans le monde de l'informatique et nous avons pu assister au prouesses accomplies par des intelligences artificielles tels que la victoire d'AlphaGo face à Lee Sedol au jeu de Go en 2016.[4] Cet événement n'est pas sans rappeler la défaite de Kasparov contre Deep Blue en 1997 qui avait défrayé le chronique.[5]
Mais comment en sommes nous arrivés là, et quels sont les techniques qui se cachent derrière ces impressionnant résultats ?
Il y a plusieurs manière de faire du machine learning ; une des plus populaire, celle qui est utilisée pour ce travail, est le réseau de neurones artificiels. Cette technologie existe depuis bien longtemps, mais ce sont les récent progrès en terme de puissance de calcul qui ont amorcé sa montée comme technologie phare du domaine.[6]

La méthode classique pour entrainer des réseaux de neurones est la rétropropagation du gradient (back-propagation). Cette technique consiste à comparer la sorties du réseau à la sortie attendu pour une entrée donnée afin de corriger les poids du réseau.[7] L'avantage de cette technique est qu'elle va converger très rapidement vers le résultat attendu (à condition que les hyperparamètres du réseau soit adaptés au problème). Le désavantage est qu'il faut avoir à sa disposition un grand nombres d'entrées dont on connait la nature afin de pouvoir les comparer aux résultats du réseau. Par exemple il existe une base de données de chiffres écrit à la main avec leur valeur réelle sur `http://yann.lecun.com/exdb/mnist/` (70'000 entrées).

Cependant générer une telle base de données est un travail énorme, ainsi ces dernières années nous avons assisté à l'émergence de nouvelles technique ne nécessitant pas d'exemples.

Une de ces techniques, qui a été utilisée pour l'apprentissage d'AlphaGo[8], est le l'apprentissage par renforcement (Reinforcement learning). Cette méthode consiste à juger les décisions prises par un agent autonome en lui donnant une recompense positive ou négative afin que l'agent, au fur et à mesure des expériences trouve une stratégie optimale.[9] Cette façon de faire peut être comparé au comportement d'un enfant qui apprends à faire du vélo. Quand il tombe il va avoir mal, c'est une récompense négative. Il va donc corriger son comportement de façon à ne plus tomber. A l'inverse, quand il arrive a aller loin il va être fier et donc favoriser ce comportement, c'est une récompense positive.

Cependant il existe une méthode encore plus généraliste pour l'apprentissage : La neuroévolution. Cette méthode se base sur les algorithmes évolutionniste dont le fonctionnement est inspiré de la sélection naturelle qui a guidé l'évolution de la vie sur terre. En effet à partir d'un ensemble d'organismes que nous allons évaluer à leur capacité effectuer une tâche donnée, ce que nous appelons le *fitness* de l'organisme. Nous allons sélectionner les meilleurs d'entre eux afin de les faire se "reproduire" pour créer la génération suivante qui sera à son tour évaluée et on recommence ce processus autant de fois que nécessaire.[10]

Le but de ce travail de bachelor est d'étudier et de comparer le comportement deux algorithmes de neuroévolution en analysant la faculté de chacun à apprendre à jouer à des jeux vidéo. Cette technique à été choisie car elle permet un développement plus libre de l'IA en ne jugeant que le résultat et non pas les actions qui y mène.

2 Réseau de Neurones Artificiels

Un réseau de neurones est un modèle, inspiré du fonctionnement du cerveau humain, qui va consister en un ensemble de neurones artificiels (aussi appelés perceptrons), disposés en couches qui vont communiquer en propageant une information.[11] En effet les neurones de notre cerveau vont collecter les signaux en provenance de leurs dendrites, puis si les signaux sont assez fort envoyer une impulsion le long de leur axone vers les neurones suivant qui vont faire de même. A noter que les connexions entre deux neurones peuvent être plus ou moins fortes (le signal va donc se propager avec une intensité variable).[12]

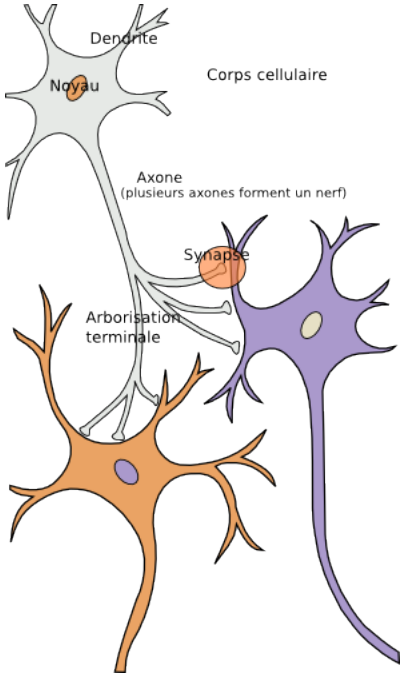

FIGURE 1 – Représentation des neurones dans le cerveau humain [1]

Les perceptrons quant à eux vont imiter (de manière simplifiée) ce comportement, chaque perceptron va prendre le signal envoyée par chacun de ses voisins de la couche précédente, puis multiplier cette valeur par le poids le connexion et finalement faire le somme de toutes les valeurs pondérées et passer cette somme dans une fonction (dites fonction d'activation) qui va placer cette somme dans un interval (entre 0 et 1 par exemple) et renvoyer le résultat de cette fonction à tous ses voisins de la couche suivante qui vont faire de même.[13]
D'un point de vue mathématique le comportement d'un perceptron peut être écrit comme :

$$o = f(\sum_{i=0}^{n} S_i * W_i) \tag{1}$$

Où
o est le signal qui va sortir du perceptron

f est la fonction d'activation
n est le nombre de voisins de la couche précédente
S est le vecteur des signaux des voisins la couche précédente
W est le vecteur des poids entre le perceptrons et ses voisins de la couche précédente

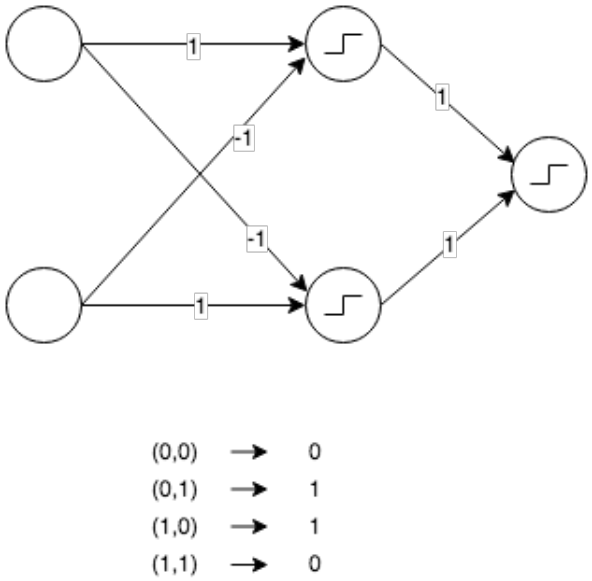

FIGURE 2 – Porte logique "ou exclusif" (XOR) simulée par un réseau de neurones

Les neurones de la première couche vont recevoir le. Ces entrées (aussi appelées *features*) dépendent du problème pour lequel le réseau est employé. Ainsi si nous cherchons à prédire le note d'un examen en fonction du temps de révision et de nombre d'heures de sommeil, les entrées du réseau seront le temps de révision et le nombres d'heures de sommeil.
Chaque neurones de la couche suivante calculera sa valeur par propagation du signal de la couche précédente afin d'arriver jusqu'à la dernière couche de neurones qui déterminera la prédiction du réseau de neurones. Cette prédiction est également appelée *label*. Dans le cas de notre exemple, le label du réseau sera la note prévue.
Avant de pouvoir utiliser notre réseau pour faire des prédictions, il faut l'entrainer, c'est à dire ajuster les paramètres (poids, topologie) afin que les labels soient le plus correct possible pour des features données.

Il est cependant important de ne pas mystifier le fonctionnement des réseaux de neurones. Ce sont certes des "boites noires", c'est à dire qu'il n'est pas possible d'analyser un réseau et d'expliquer pourquoi une connexion donnée a un tel poids. En revanche, il ne faut pas perdre de vue que leur comportement est exprimable par une fonction mathématique. En effet tout comme le comportement d'un perceptron est décrit dans l'équation 1 le comportement de n'importe quel réseau de neurones peut-être décrit par une equation dépendant du réseau.

2.1 Apprentissage supervisé

L'apprentissage supervisé est une manière d'entrainer un réseau de neurones avec un ensemble de couples (features, labels). Le but est de commencer avec un réseau initialisé aléatoirement, puis de lui donner en entrée des features dont on connait les labels et de comparer les labels donnés par le réseaux aux labels attendus.
En fonction du taux d'erreur, il sera possible de corriger les poids des connexions du réseau pour minimiser ce taux.

Les avantages principaux de cette technique d'apprentissage sont la rapidité d'entrainement et la justesse atteignable étant donné que le fonctionnement ne dépend pas d'heuristiques.

Les désavantages sont le temps de génération de la base d'exemples et le fait qu'il faille connaitre le label attendu pour chaque feature (nous savons à coup sur qu'une image de chien doit être classifiée comme "chien" mais nous ne savons pas le coup idéal à jouer selon l'état d'un plateau de jeu).

2.2 Apprentissage non supervisé

2.3 Apprentissage par renforcement

2.4 Neuroevolution

2.5 Fonctions d'activation

Les fonction d'activations sont des fonctions qui ont pour tâche de limiter la valeur des neurones dans un interval donné. Afin de pouvoir appliquer la rétropropagation du gradient il faut également que la fonction soit dérivable. Voici quelques exemples de fonctions utilisées couramment :

La sigmoid $\sigma : \mathbb{R} \to]0, 1[$

$$\sigma : x \mapsto \frac{1}{1 + e^{-\beta x}} \tag{2}$$

La tangente hyperbolique $tanh : \mathbb{R} \to]-1, 1[$

$$tanh : x \mapsto \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3}$$

La fonction step $step : \mathbb{R} \to [0, 1]$

$$step : x \mapsto \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{otherwise} \end{cases} \tag{4}$$

2.6 Types de réseau

2.6.1 Feed-Forward

2.6.2 Long Short Term Memory

2.6.3 Convolutional Neural Networks

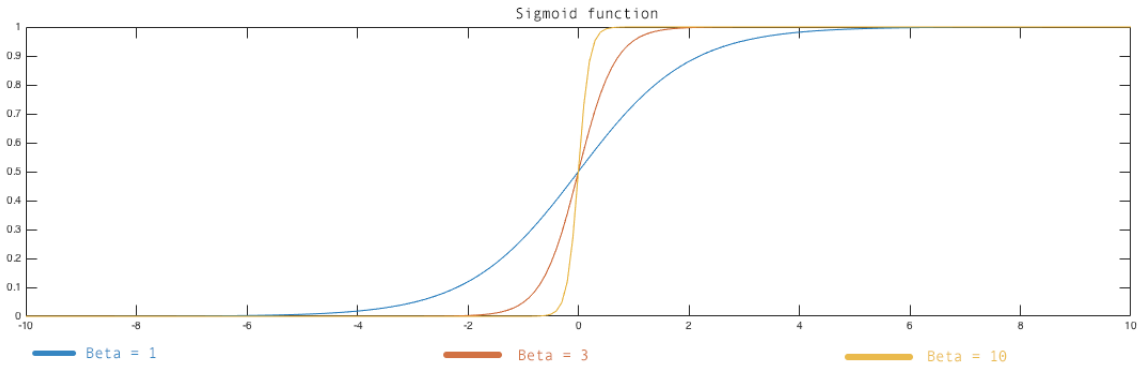

FIGURE 3 – Fonction Sigmoid avec des β variables

3 Agent Intelligent

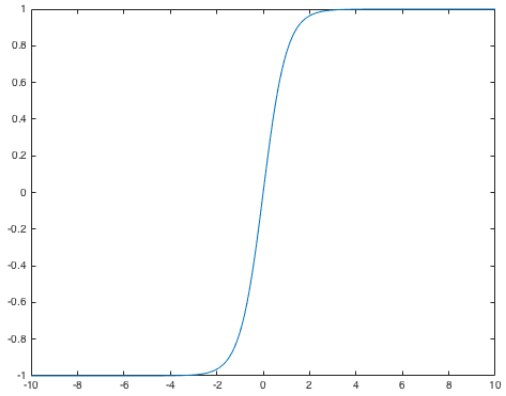

FIGURE 4 – Fonction tangente hyperbolique

4 Algorithmes évolutionnistes

Un algorithme évolutionniste est un algorithme d'optimisation métaheuristique qui cherche, par un processus inspiré de la sélection naturelle, une solution optimale à un problème. Ces algorithmes utilisent des iterations stochastiques pour arriver à maximiser une fonction évaluant leurs performances, dite fonction de fitness.[14]
Il existe plusieurs sortes d'algorithmes évolutionniste, la neuroévolution en est une et c'est celle-ci qui va nous intéresser. Le but de cette technique est d'utiliser les principes des algorithmes évolutionnistes afin d'optimiser un réseau de neurones pour effectuer la tâche désirée.
Les paramètres sur lesquels l'algorithme va intervenir peuvent être les poids des connexions dans le réseau ainsi que la topologie du réseau (nombre de neurones, connexions entre chaque neurones, nombre de couches).

4.1 Génome & Phénome

Le génome est l'ensemble du matériel génétique d'un individu, celui-ci contient toutes les informations nécessaire à la création de l'individu.[15] Chez les humain le génome est organisé en 23 paires de chromosomes. A partir de ces chromosomes, qui sont uniques à chaque organismes, on peut créer le phénome.
Le phénome est l'ensemble de tout les traits observables chez un individu (p.ex La couleur des yeux, de la peau, des cheveux etc...).[16]

Dans le cadre de la neuroévolution le génome est un encodage permettant la création du phénome qui est le réseau de neurones.

4.2 Population initiale

La base d'un algorithme évolutionniste est la population initiale, celle-ci doit être générée aléatoirement afin de représenter un vaste spectre de possibilités. Nous ne retrouverons dans cette population aléatoire aucuns

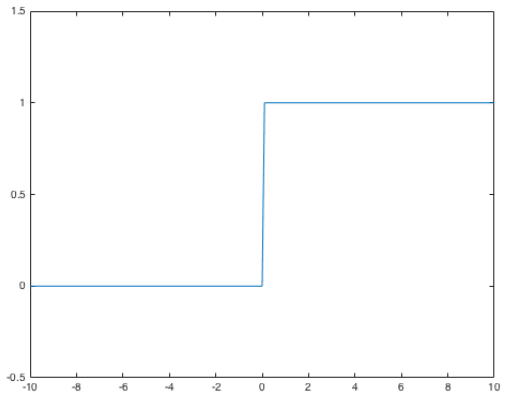

FIGURE 5 – Fonction step

individu capable d'accomplir parfaitement la tâche demandée, mais certains aurons des comportements qui les aiderons a d'obtenir un fitness plus élevé que leurs voisins, ceux-ci vont donc passer leurs caractéristiques à la génération suivante.

4.3 Sélection

La sélection est un concept essentiel dans l'utilisation d'algorithmes évolutionnistes. L'objectif est de choisir, en prenant en compte le fitness afin que les individus performant soit plus souvent sélectionnés pour être parents. Ainsi, les caractéristiques qui les rendent plus performant seront plus largement passés vers la génération suivante alors que les caractéristiques des individus plus faibles disparaîtront rapidement.[2]

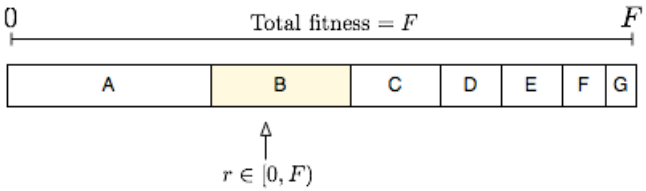

FIGURE 6 – Exemple de sélection proportionnelle au fitness[2]

4.4 Crossover

Un crossover (où enjambement en français) est une opération génétique qui croise les gènes de deux parents afin de créer le gène de l'enfant, ce dernier va donc hériter de certaines caractéristiques de l'un où l'autre parent. Un exemple visible de cette opération chez les humains est que l'on reconnait des traits (couleur de peau, yeux, cheveux) des parents chez les enfants.[17]

4.5 Mutation

Les mutations sont des événements aléatoires qui vont altérer le génome de façon à créer une innovation, qui va par exemple se traduire par un comportement différent du génome. Dans certains cas les mutations seront bénéfiques (p.ex La capacité à respirer hors de l'eau), dans d'autre la mutation causera un comportement désavantageux (p.ex Malformation des membres).[18]

4.6 Elitisme

L'Elitisme est un concept qui va permettre la survie des meilleurs individus d'une génération vers la génération suivante sans que leur code génétique soit modifié.[19]
La raison pour laquelle ce concept est mis en place est que les solutions optimales trouvées ne doivent pas être perdues à cause de mutations désavantageuses.

5 Evolution du perceptron multicouche - Algorithme Naïf

Cette section détaille le fonctionnement de l'algorithme évolutionniste mis en place pour faire évoluer les perceptrons multicouche à topologie fixe. Celui-ci va donc uniquement faire varier les poids des connexion dans le réseau pour optimiser son comportement. L'algorithme à été créé en s'inspirant des concepts d'algorithmes évolutionnistes décrit précédemment.

5.1 Génome & Phénome

Pour cet algorithme le génome est simplement la liste des poids constituant le réseau de neurones. Ainsi, étant donne que la topologie du réseau est fixe, on peut le reproduire à l'identique en assignant les bons poids.

Le phénome dans le cas de cet algorithme sera une réseau de neurones à topologie fixe et entièrement connecté (Chaque neurones d'une couche donnée est connecté a chaque neurones de la couche suivante).

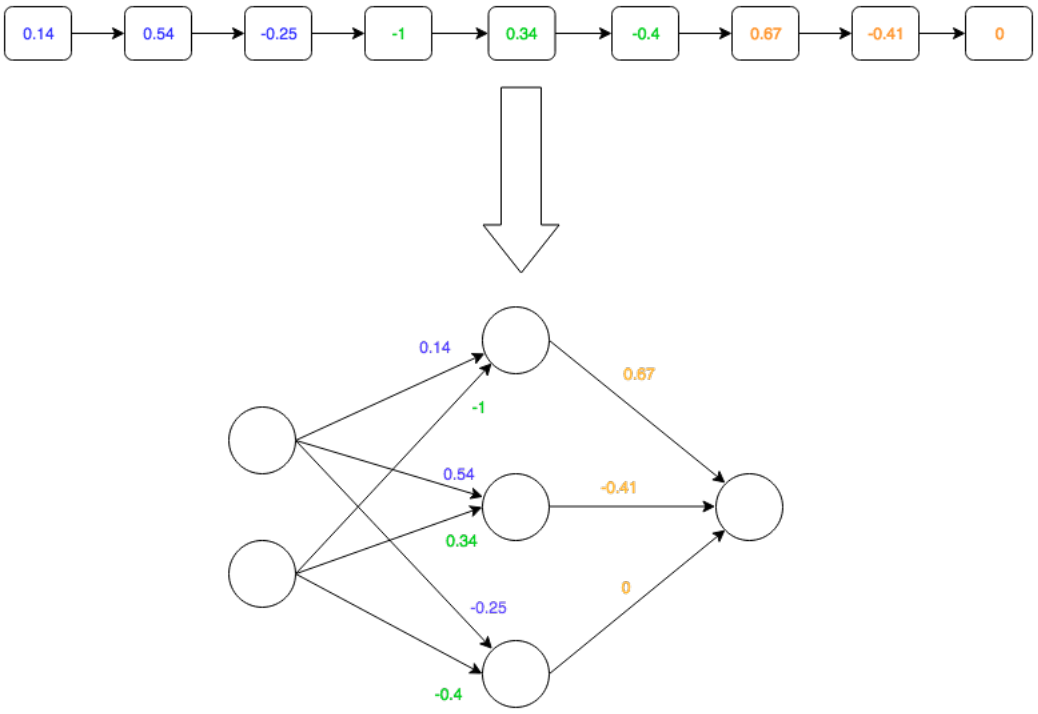

FIGURE 7 – Passage du génome au phénome

5.2 Population initiale

La population initiale est un ensemble de perceptrons multicouche dont la topologie est la même, les poids entre les couches sont assignés aléatoirement à une valeur entre -1 et +1 afin de représenter un spectre de possibilités aussi large que possible.

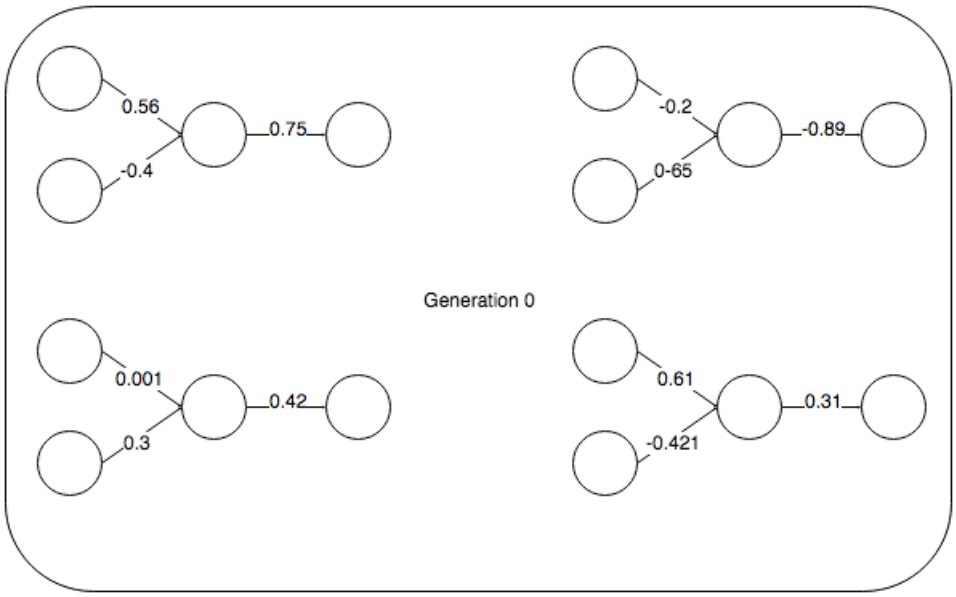

FIGURE 8 – Exemple de population initiale

5.3 Sélection

La sélection est utilisée pour choisir un individu de la génération précédente qui va soit passer ses gènes directement (être cloné), soit faire un crossover avec les gènes d'un autre individu lui aussi sélectionné. La probabilité de l'une ou l'autre de ces actions est de 50%.
Le génome résultant de cette sélection subira ensuite une potentielle mutation. Ce processus est répété autant de fois que nécessaire pour créer une nouvelle génération contenant autant d'organismes que la précédente.

5.4 Crossover

Le crossover fonctionne d'après un principe très simple :

- Choisir un point de croisement
- Assigner chez l'enfant tout les gènes qui précèdent ce point depuis le premier parent
- Assigner chez l'enfant tout les gènes qui suivent ce point depuis le second parent

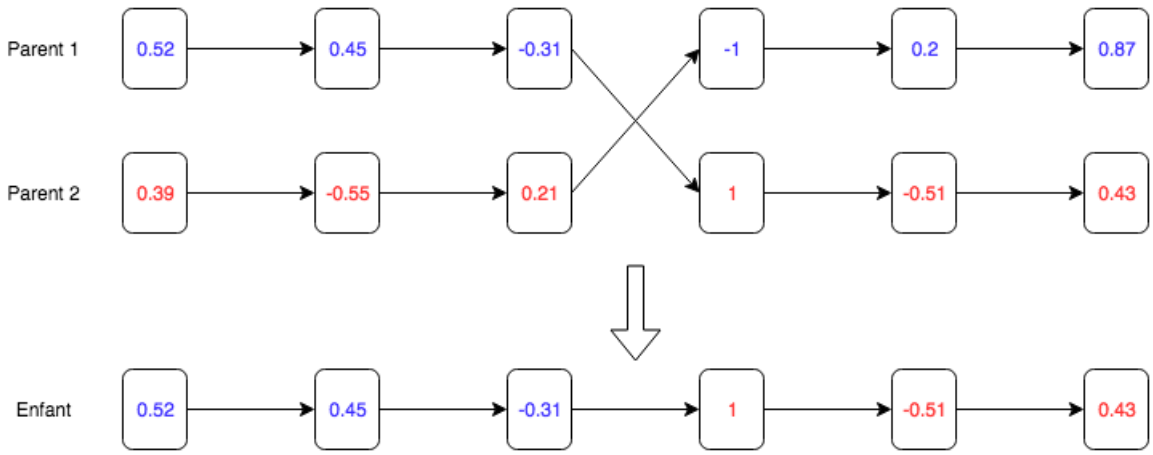

FIGURE 9 – Fonctionnement d'un crossover

Ainsi, les traits permettant une meilleure survie vont rapidement se répandre dans la population car les individus ayant obtenu un meilleur fitness serons plus souvent sélectionnés pour faire une crossover avec un autre individu.

5.5 Mutation

Dans le cas de cet algorithme, étant donné que la topologie est fixe, la mutation sera simplement une variation aléatoire d'un poids dans le réseau de neurones. Ainsi chaque enfant de chaque génération aura 10% de chance que l'un de ses gènes subisse une mutation.
Le poids subira donc une variation d'une valeur aléatoire choisie entre +0.5 et -0.5 cependant le poids sera ensuite fixée dans l'interval $[-1, 1]$.

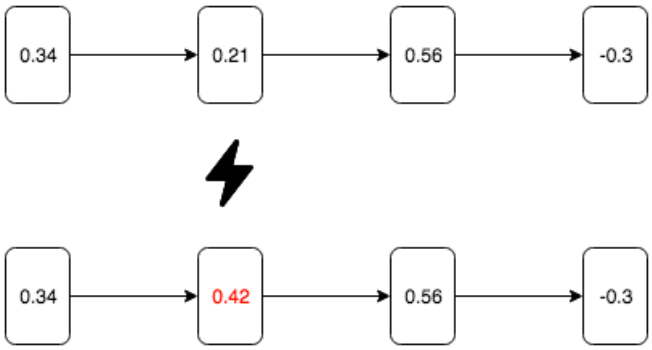

FIGURE 10 – Exemple de mutation dans un génome

5.6 Elitisme

Pour cet algorithme, seul le champion de la génération (organisme ayant obtenu le meilleur fitness) est préservé sans altération de son code génétique pour la génération suivante.

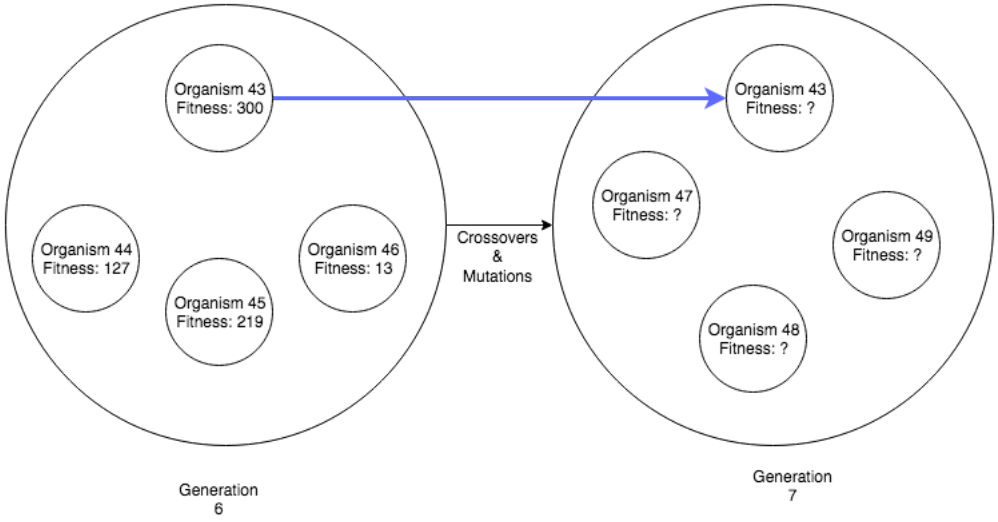

FIGURE 11 – Elitisme d'une génération vers la suivante

5.7 Déroulement de l'algorithme

L'organigramme ci-dessous décrit le comportement global de l'algorithme. La fonction random(0, 1) va tirer un nombre aléatoire entre compris dans l'interval $[0, 1[$.

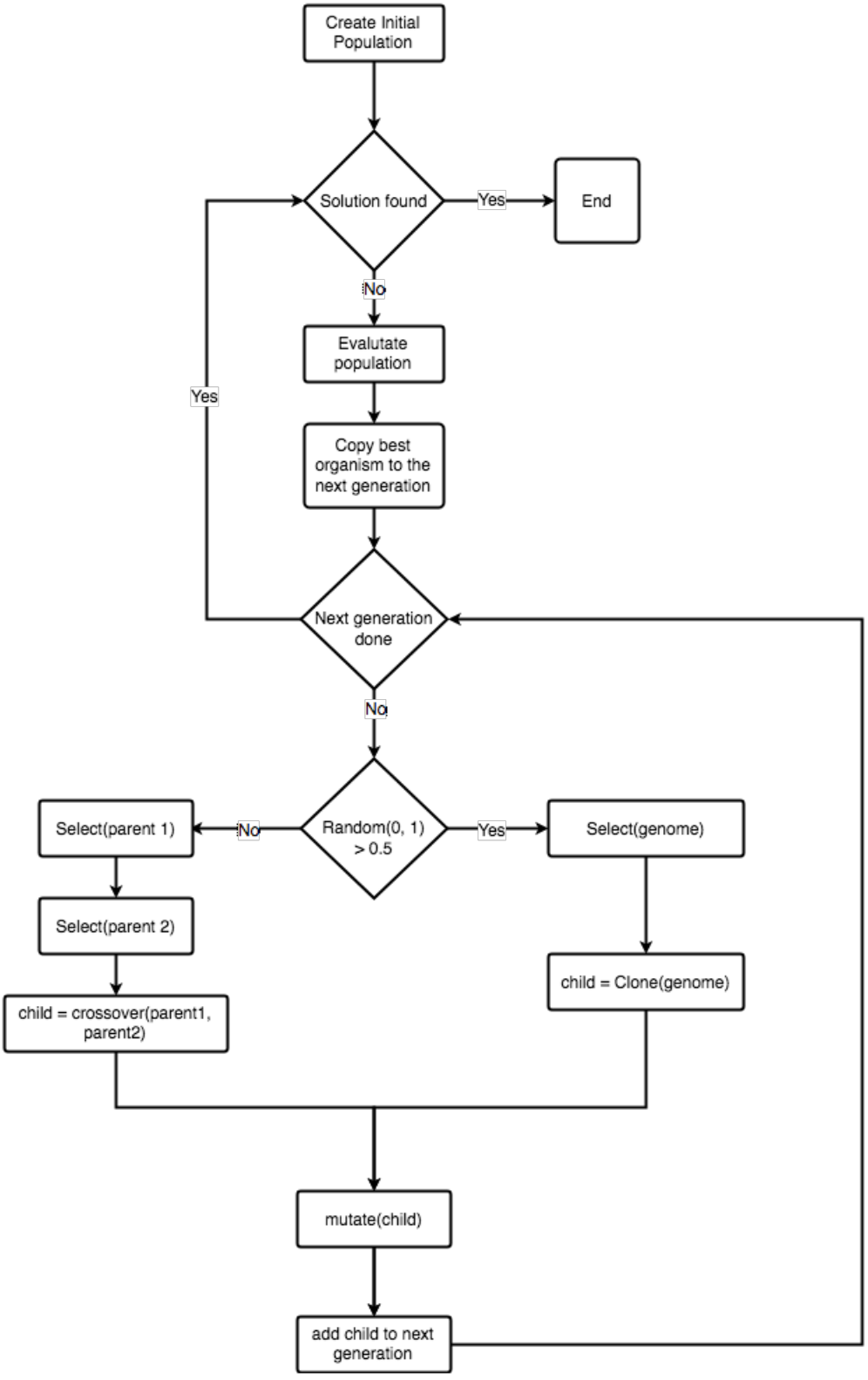

FIGURE 12 – Organigramme de l'algorithme naïf

6 NEAT

L'autre algorithme de neuroévolution mis en place est le NEAT, créé par Ken Stanley à Université du Texas à Austin. Cet algorithme détaille comment obtenir des réseaux de neurones dont la topologie change au fil des évolutions.
Il apporte trois techniques essentielles à son fonctionnement : suivre l'évolution des gènes avec un historique pour permettre les crossover parmi les différentes topologies, séparer les organismes en espèces afin de préserver l'innovation et faire évoluer les topologies de manière incrémentale en partant de structures simples afin d'obtenir des résultats minimaux.[20]

6.1 Génome & Phénome

Dans le cas de NEAT, le génome est plus complexe, en effet il doit représenter tout les neurones et toutes les connexions.

6.2 Population initiale

La population initiale doit être aussi simple que possible, cependant il n'y a pas de règles dictant la manière dont elle doit être faite. Pour cet algorithme, la population initiale est donc un ensemble de perceptron multicouche avec seulement une couche d'entrée et une couche de sortie dans lequel chaque entrée est reliée à chaque sorties.

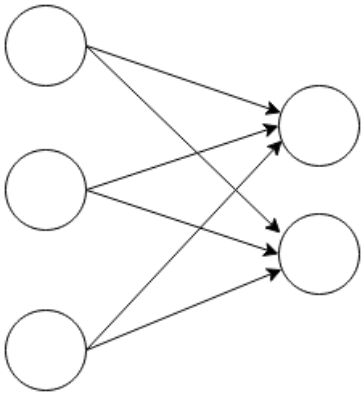

FIGURE 13 – Exemple de population initiale dans NEAT

Ce choix à été fait pour conserver un équilibre entre une population simple (pas de couche intermédiaire) mais qui soit rapidement capable de s'ajuster car les connexions sont déjà disponibles, on ne pert donc pas de générations uniquement pour créer les connexions.

6.3 Spéciation

6.4 Sélection

6.5 Crossover

Le crossover dans le cadre du NEAT pose le problème des conventions concurrentes (competing conventions). En effet il possible que deux réseaux produisent le même résultat mais de manière différente, le risque est que lorsque nous croisons les deux réseau nous perdions une partie de l'information qui les rends efficace.

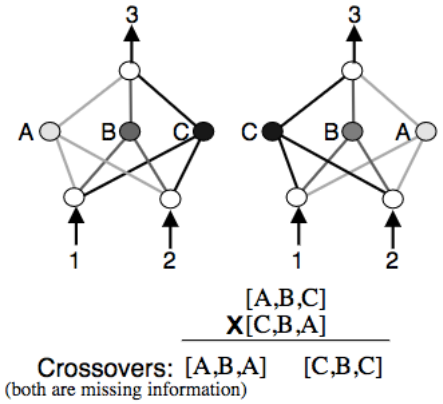

FIGURE 14 – Problème des conventions concurrentes, les deux réseaux atteignent le même résultat mais leurs possibles enfants en n'aurons que 2/3 de l'information [3]

Afin de pallier à ce problème, nous suivons l'apparitions d'innovations,

6.6 Mutation

6.7 Elitisme

7 Architecture Réseau

Afin d'optimiser la vitesse de calcul, un système distribué à été mis en place. Celui-ci repose sur une architecture client-serveur classique où le client effectue les simulations et le serveur gère les génomes et s'occupe de distribuer de manière équitable le travail entre tous les clients.

7.1 Protcole

Le protocole à été établi à l'aide du langage protobuf qui permet d'avoir une définition claire ne dépendant pas des langages dans lesquels le protcole est ensuite implémenté.
Voici le fichier qui défini le protocole :

Protocole

```
syntax = "proto2";
option java_package = "me.pv.mg.protobuf";

enum MGMessages {
    MG_JOIN = 1;
    MG_JOIN_RESPONSE = 2;
    MG_COMPUTE_REQUEST = 3;
    MG_COMPUTE_RESPONSE = 4;
    MG_COMPUTE_RESULT = 5;
    MG_END = 6;
}

enum MGNetworkType {
    MG_MULTILAYER_PERCEPTRON = 1;
    MG_NEAT = 2;
}

message MGJoin {
    optional string pretty_name = 1;
    optional bool spectator = 2;
}

message MGJoinResponse {
    required bool accepted = 1;
    optional string reason = 2;
}

message MGComputeInfo {
    required string game = 1;
    required .MGNetworkType net_type = 3;
    required string net_metadata = 4;
}
```

```
message MGComputeRequest {
    required .MGComputeInfo compute_info = 1;
    required string genome = 2;
}

message MGComputeResponse {
    required bool can_do = 1;
}

message MGComputeResult {
    required float fitness = 1;
    optional uint32 time = 2;
}

message MGEnd {
    optional string message = 1;
}
```

Le protocole définit donc 6 types de messages :

- MG_JOIN, Envoyé par le client, c'est une demande à rejoindre le groupe de calcul. En paramètres sont donnés le nom du client et si il veut rejoindre en tant que specateur ou non (fonctionnalité expliquée dans la partie client).
- MG_JOIN_RESPONSE, Envoyé par le serveur, confirme ou infirme l'ajout au groupe du client. Il n'existe pour l'instant aucune raison pour le rejet d'un client.
- MG_COMPUTE_REQUEST, Envoyé par le serveur, demande au client de calculer le fitness d'un génome donné sur un jeu donné.
- MG_COMPUTE_RESPONSE, Envoyé par le client, indique au serveur si oui ou non le client est en mesure d'effectuer la simulation.
- MG_COMPUTE_RESULT, Envoyé par le client, donne le fitness obtenu par le génome donné dans le message MG_COMPUTE_REQUEST sur le jeu donné dans ce même message et le temps (ms) pris par le client pour effectuer la simulation.
- MG_END, Envoyé par n'importe quel entité, indique un désir de terminer la connexion.

7.2 Serveur

Comme écrit ci-dessus, le serveur à la lourde tâche de gérer tous les génomes et leurs fitness afin de mettre en oeuvre les algorithmes évolutionniste qui vont permettre l'évolution. En plus de cela il va également devoir servir les pages web servant à la gestion et à la surveillance du processus évolutif.
Afin de pouvoir effectuer toutes ces tâches, le serveur est constitué de modules node.js qui vont chacun gérer une partie de ce qui est lui est demandé.
L'architecture se présente ainsi :

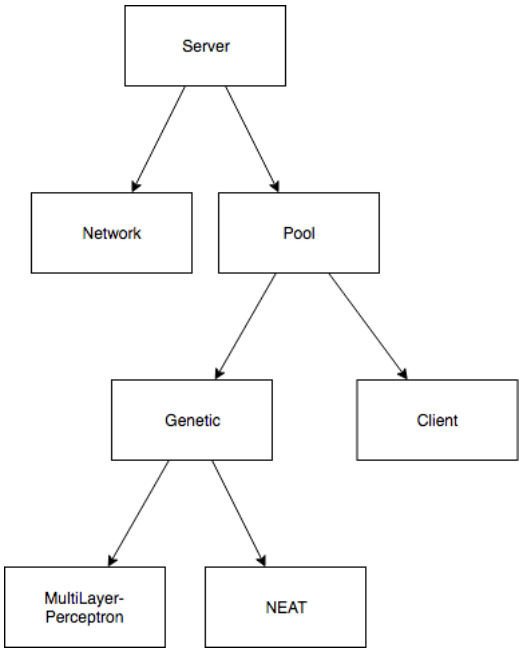

FIGURE 15 – Architecture des modules du serveur

7.3 Client

Le client va devoir effectuer la simulation demandée par le serveur, avec le bon réseau de neurones

7.3.1 Réseaux de neurones

7.3.2 Simulateur

7.3.3 Tests

8 Interface de Controle

8.1 Mode

8.2 Tâche

8.3 Ouvriers

8.4 Graphs

9 Base de donnés

10 API

11 Jeux

11.1 Asteroid

11.1.1 Principe

11.1.2 Entrées

11.1.3 Sorties

11.1.4 Résultats

12 Conclusion

Références

[1] Fernandes. Exosquelette et bionique, le système nerveux, 2013. [Online ; accessed 30-June-2018].

[2] Wikipedia contributors. Fitness proportionate selection — Wikipedia, the free encyclopedia, 2018. [Online ; accessed 4-July-2018].

[3] Kenneth Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *The MIT Press*, page 6, 2002.

[4] Wikipedia contributors. Alphago versus lee sedol — Wikipedia, the free encyclopedia, 2018. [Online ; accessed 3-July-2018].

[5] Mark Robert Anderson. Twenty years on from deep blue vs kasparov : how a chess match started the big data revolution. *The Conversation*, 2017.

[6] Robert D. Hof. With massive amounts of computational power, machines can now recognize objects and translate speech in real time. artificial intelligence is finally getting smart. *The Conversation*.

[7] Raúl Rojas. Neural networks, 1996.

[8] David Silver and Demis Hassabis. Alphago : Mastering the ancient game of go with machine learning. *Google AI Blog*, 2016.

[9] Wikipedia contributors. Reinforcement learning — Wikipedia, the free encyclopedia, 2018. [Online ; accessed 3-July-2018].

[10] Wikipedia contributors. Neuroevolution — Wikipedia, the free encyclopedia, 2018. [Online ; accessed 22-April-2018].

[11] Wikipedia contributors. Artificial neural network — Wikipedia, the free encyclopedia, 2018. [Online ; accessed 30-June-2018].

[12] Stephen Williams and Pankaj Sah. Brain functions, 2018. [Online ; accessed 21-April-2018].

[13] Wikipedia contributors. Perceptron — Wikipedia, the free encyclopedia, 2018. [Online ; accessed 3-July-2018].

[14] Wikipedia contributors. Evolutionary algorithm — Wikipedia, the free encyclopedia, 2018. [Online ; accessed 4-July-2018].

[15] Wikipedia contributors. Genome — Wikipedia, the free encyclopedia, 2018. [Online ; accessed 4-July-2018].

[16] Wikipedia contributors. Phenome — Wikipedia, the free encyclopedia, 2018. [Online ; accessed 4-July-2018].

[17] Wikipedia contributors. Chromosomal crossover — Wikipedia, the free encyclopedia, 2018. [Online ; accessed 4-July-2018].

[18] Wikipedia contributors. Mutation — Wikipedia, the free encyclopedia, 2018. [Online ; accessed 4-July-2018].

[19] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm, 1995.

[20] Wikipedia contributors. Neuroevolution of augmenting topologies — Wikipedia, the free encyclopedia, 2018. [Online ; accessed 30-June-2018].