

Projet IA

LEARNING

PAC-MAN



Leleux Laurent & Moulart Kévin

LEARNING

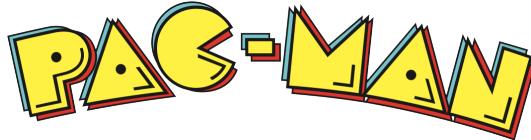


Table des matières

I. Introduction	3
II. Algorithmes d'apprentissage	4
a. Q-Learning.....	4
b. Réseau de neurones	6
c. Algorithme génétique	9
III. Choix d'un algorithme : l'algorithme génétique	12
IV. Adaptation de l'algorithme choisi au problème	13
V. Mode d'emploi pour l'intégration au projet Ms. Pac-man	16
VI. Analyse des résultats	17
VII. Conclusion	24
VIII. Bibliographie.....	25

I. Introduction

Dans le cadre du projet d'intelligence artificielle, il nous a été demandé de mettre en pratique une technique de programmation permettant à une intelligence artificielle d'apprendre une stratégie pour contrôler au mieux soit Ms. Pac-man, soit ses quatre ennemis. Le choix nous était donné quant à l'algorithme à utiliser pour réaliser cet apprentissage :

- Le Q-Learning
- Le réseau de neurones
- L'algorithme génétique

Nous détaillerons le fonctionnement spécifique de chacun de ces algorithmes ci-après, mais attardons-nous sur cette notion d'apprentissage, et d'abord sur la notion d'intelligence artificielle.

Marvin Lee Minsky, l'un des pères fondateurs de l'intelligence artificielle la décrivait comme « *the building of computer programs which perform tasks which are, for the moment, performed in a more satisfactory way by humans because they require high level mental processes such as: perception learning, memory organization and critical reasoning* ». C'est-à-dire que l'on considère qu'un programme sera intelligent s'il accomplit des tâches jusque là effectuées par des humains car elles requièrent un processus mental de haut niveau. Par exemple, le fait de jouer à un jeu est typiquement attribué à un processus décisionnel que l'humain est plus à même de réaliser que la machine (cfr. les parties de jeu vidéo en live contre des autres joueurs, plutôt qu'en local contre des bots, qui ont beaucoup plus de succès que ces dernières !).

Le problème que nous avons à résoudre en faisant jouer Ms. Pac-man ou les fantômes relève donc bien du domaine de l'intelligence artificielle de ce point de vue-là. Mais il va également plus loin : nous ne devons pas créer une intelligence en la rendant nous-même intelligente et en lui codant des réactions à des stimuli externes pour lui faire jouer de belles parties, mais bien lui donner toutes les compétences nécessaires pour qu'elle **apprenne** elle-même comment jouer de telles parties.

Et la véritable intelligence de notre programme se trouve bel et bien là : dans le fait que nous ne disons pas à Ms. Pac-man d'éviter les fantômes ou de manger les pastilles, ni aux dits fantômes de manger Ms. Pac-man et de la fuir quand ils sont vulnérables, ils **l'apprennent** eux-mêmes et trouvent même des stratégies pour devenir performants dans la tâche demandée !

Nous allons maintenant vous exposer les différents algorithmes, leurs forces et leurs faiblesses, mais surtout leur fonctionnement afin de vous faire comprendre notre choix de l'algorithme génétique comme solution au problème posé.

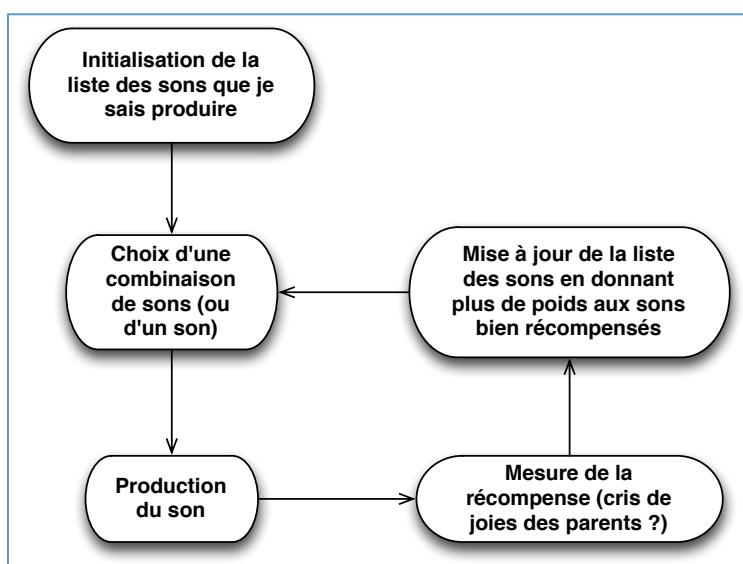
Bonne lecture

II. Algorithmes d'apprentissage

a. Q-Learning

La technique du Q-Learning est une technique d'apprentissage par renforcement non supervisé. En clair, cela vise à créer un agent logiciel de contrôle pratiquement optimal pour un problème donné.

Le principe de fonctionnement de cet algorithme est assimilable à celui d'apprentissage de la parole par un enfant : des expériences au hasard, et une récompense plus ou moins importante selon la réussite de l'expérience, qui est mémorisée. L'enjeu est donc de retrouver les expériences les plus récompensées et de les reproduire au mieux en variant des paramètres.



Pour illustrer cette technique, revenons au parallèle avec un bébé qui apprend à parler : il commence par gazouiller des « areuh, beuh, ... », ce qui n'entraîne aucune récompense des parents, tout juste contents que les cordes vocales de leur enfant soient fonctionnelles (quoique, à trois heures du matin, ils sont moins heureux de l'apprendre...). Alors il persévère, il essaye des variations jusqu'à atteindre le « papa » ou « maman » tant attendu, aussitôt récompensé par des cris de joie et des rires : grosse récompense. Le bébé mémorise alors ce résultat qui vient « renforcer » sa connaissance du

langage, et peu à peu, expérience après expérience, il évolue, d'une stratégie aléatoire où il enchaîne les sons au hasard, à une stratégie contrôlée où il les enchaîne pour créer des mots.

Cet exemple est simplifié évidemment, l'apprentissage du langage n'est pas qu'une question de renforcement, mais il illustre bien ce principe d'apprentissage.

Nous avons parlé de processus à renforcement, expliqué ci-dessus, non supervisé. Cela signifie que l'on ne fournit pas au programme une masse de données à digérer pour apprendre, contrairement aux réseaux de neurones qui apprennent par l'exemple et nécessitent une supervision pour vérifier qu'ils digèrent correctement l'information, ici l'apprentissage se fera par essais/erreurs/récompenses.

Cet algorithme est particulièrement facile à mettre en place, le cœur de sa méthode ne nécessitant que quelques lignes de codes, mais sa simplicité n'a d'égal que la difficulté de paramétrage qui s'ensuit.

Analytiquement parlant, le modèle du problème est résumé par un agent, dans un certain état (S) et un set d'actions par état (A). En exécutant une action $a \in A$, l'agent peut passer d'un état à un autre. Chaque état donnera à l'individu une récompense (un entier ou un réel). Le but de l'agent sera de maximiser la récompense totale. Il réalisera cela en apprenant quelle action est optimale pour chaque état. Nous pouvons donc trouver la fonction décrivant la qualité d'une paire état-action :

$$Q : S \times A \rightarrow \mathbb{R}$$

Avant que l'apprentissage ait commencé, Q renvoie une valeur fixe, choisie par le programmeur. Ensuite, à chaque fois que l'individu recevra une récompense (l'état change), de nouvelles valeurs sont calculées pour chaque combinaison d'un état $s \in S$, et d'une action $a \in A$. Le cœur de l'algorithme est donc une simple mise à jour de la valeur à chaque itération, prenant en compte l'ancienne valeur et l'ajustant sur base de la nouvelle information.

$$Q(s_t, a_t) \leftarrow \overbrace{Q(s_t, a_t)}^{\text{ancienne valeur}} + \overbrace{\alpha_t(s_t, a_t)}^{\text{taux d'apprentissage}} \times \left[\underbrace{\frac{R(s_t)}{\text{récompense}} + \gamma}_{\text{facteur d'actualisation}} \cdot \underbrace{\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})}_{\text{max future valeur}} - \underbrace{Q(s_t, a_t)}_{\text{ancienne valeur}} \right]$$

Où $R(s_t)$ est la récompense observée depuis s_t , et $\alpha_t(s_t, a_t)$ ($0 < \alpha \leq 1$) est le taux d'apprentissage (qui peut être le même pour toutes les paires). Le facteur d'actualisation γ est tel que $0 \leq \gamma < 1$.

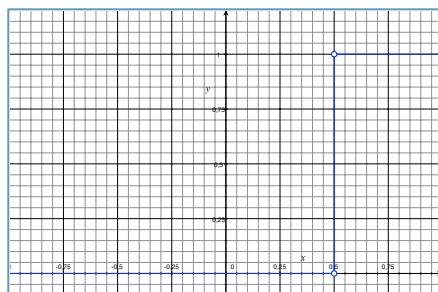
L'algorithme s'arrête quant l'état s_{t+1} est un état final (ou absorbant).

Notons que pour de tels états s_f , $Q(s_f, a)$ n'est jamais mis à jour et donc reste à la valeur initiale.

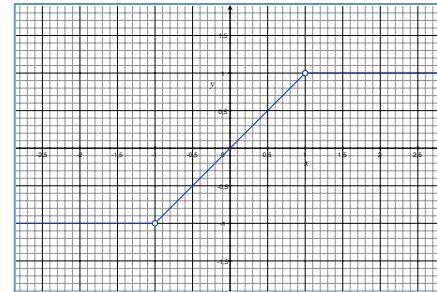
b. Réseau de neurones

Quand on entend parler de réseaux de neurones, une image nous vient directement en tête, celle d'un cerveau humain. Il semble logique que dans le but de recréer des facultés typiquement humaines, on se tourne vers ce qui fait l'intelligence de l'humain : ces milliards de petites cellules interconnectées qui, ensemble, nous permettent les plus grandes prouesses. Mais encore faut-il adapter l'idée du neurone biologique à une modélisation formelle qui permettra son implémentation fonctionnelle.

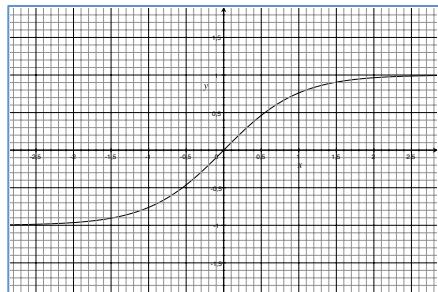
Le réseau doit donc être composé de neurones, intéressons nous d'abord à ceux-ci : chaque neurone effectue le même travail : il prend en entrée des données et suivant le poids attribué à leur source (ω), c'est à dire la fiabilité respective de chaque source, et la valeur reçue de celle-ci (x), calcule la combinaison linéaire des différentes données reçues : $\sum_{i=1}^n \omega_i x_i$. Le neurone lui applique ensuite une fonction dite « fonction d'activation» visant à renvoyer une valeur de réponse bornée. Voici des exemples de fonctions d'activation:



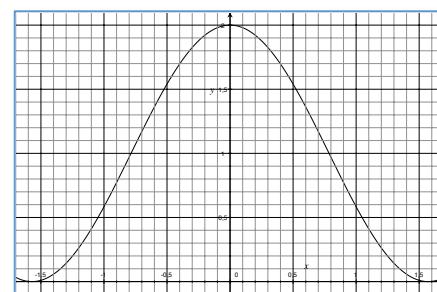
Fonction seuil



Fonction linéaire par morceaux

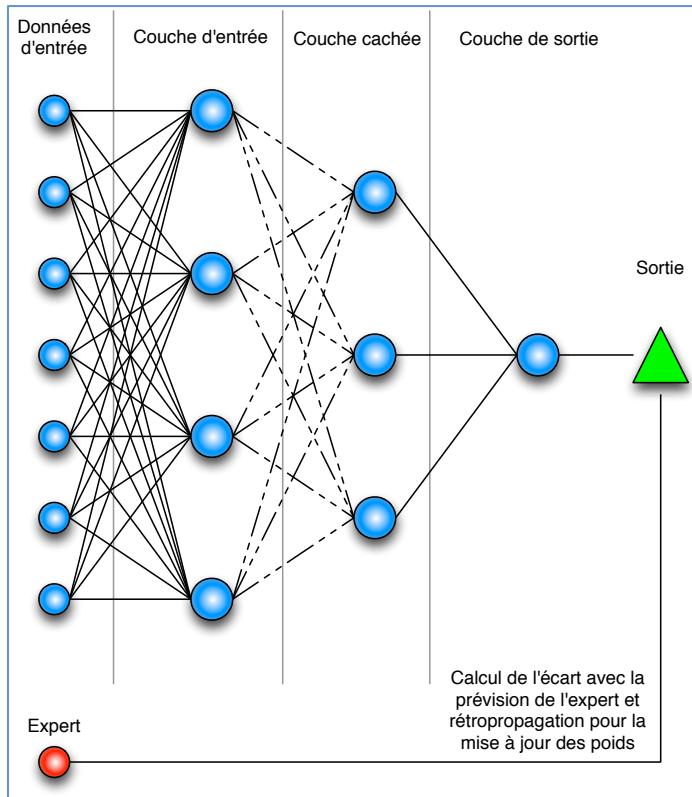


Fonction sigmoïde



Gaussienne

Ces neurones sont répartis en couches, avec une couche d'entrée, chargée de récupérer les informations en direct, puis une ou des couche(s) cachée(s), composée(s) d'un nombre variable de niveaux. Chaque niveau est constitué d'un nombre variable de neurones. Chaque neurone de chaque couche est unique par la fonction d'activation qu'il utilise et par les poids accordés à chacune de ses sources de données. Néanmoins, tous font le même travail sur les mêmes données sources et transmettent ensuite leurs données transformées à la couche suivante, qui s'en servira à son tour comme source.



On procède ainsi jusqu'à la couche de sortie, qui rassemble l'information et restitue la réponse finale du réseau de neurones. Cette réponse est alors comparée avec celle de l'expert (ou du jeu de tests d'apprentissage). La différence entre les réponses indique la performance du réseau. Elle est alors répercute dans le sens inverse de l'information sur le réseau de neurones comme expliqué ci-après. Ce comportement est appelé rétro propagation.

Afin d'expliquer ce mécanisme de rétro propagation, il nous faut comprendre l'entièreté d'un cycle d'apprentissage, depuis le tout début, c'est à dire depuis la mise en route et les premières données reçues.

Pratiquement, lors de la mise en place du réseau, on détermine le nombre de couches et le nombre de neurones par couche. Les poids des connexions sont alors attribués aléatoirement (en général de petites valeurs, par exemple entre -0.1 et 0.1). On présente enfin au réseau un jeu de données d'apprentissage, pour lequel les valeurs cibles que le réseau de neurones doit apprendre à prédire sont connues.

L'algorithme se déroule alors comme suit :

1. Nous présentons à l'entrée du réseau un échantillon \vec{x} , auquel correspond un échantillon de valeurs cibles \vec{t} .

2. Nous propageons le signal en avant dans les couches du réseau de neurones :

$$x_k^{(n-1)} \mapsto x_j^{(n)}$$

3. Cette propagation se fait à l'aide de la fonction d'activation f et de la combinaison linéaire des poids et des entrées que nous noterons h , ainsi que de ces poids $\vec{\omega}_{jk}$ entre les neurones $x_k^{(n-1)}$ et $x_j^{(n)}$, de k vers j .

$$x_j^{(n)} = f^{(n)}(h^{(n)}) = f^{(n)}\left(\sum_k \omega_{jk}^{(n)} x_k^{(n-1)}\right)$$

4. Une fois chaque couche traversée, nous obtenons une sortie calculée \vec{y} .

5. Nous calculons alors l'erreur entre la sortie obtenue \vec{y} et la sortie attendue \vec{t} pour cet échantillon, pour chaque neurone i de la couche de sortie :

$$e_i^{\text{sortie}} = f'(h_i^{\text{sortie}})[t_i - y_i] \quad (\text{Où } f' \text{ est la dérivée de } f)$$

6. Nous propageons ensuite l'erreur calculée vers l'arrière $e_i^{(n)} \mapsto e_j^{(n-1)}$ de la manière suivante :

$$e_j^{(n-1)} = f'^{(n-1)}(h_j^{(n-1)}) \sum_i \omega_{ij} e_i^{(n)}$$

7. Et enfin, nous mettons à jour les poids dans chaque couche :

$$\Delta \omega_{ij}^{(n)} = \lambda e_i^{(n)} x_j^{(n-1)}$$

Où λ représente le taux d'apprentissage (en général faible, $\lambda < 1$).

Nous constatons donc aisément que les réseaux de neurones font de l'apprentissage dit supervisé : ils *observent* un jeu de données, et comparant à chaque fois leurs prévisions avec les valeurs correctes, ce qui leur permet ensuite de stocker la connaissance empirique et de la rendre disponible à l'usage.

L'information est stockée dans les connexions entre les neurones, et dans les poids accordés à ces connexions.

c. Algorithme génétique

L'algorithme génétique est une adaptation mathématique et logique de la théorie Darwinienne de l'évolution : la survie du plus apte. Celle-ci a donc pour but annoncé de trouver « le plus apte », il s'agit ici d'optimiser un comportement pour trouver un individu optimal dans un système donné. Pour arriver à cet individu, Il faut d'abord traduire les traits de caractères qui le définissent, lui et ses semblables afin de les encoder dans des chromosomes, juste comme dans la réalité de la théorie Darwinienne.

Tout d'abord, un petit exemple sur la théorie de l'évolution qui aidera à comprendre mieux la suite. Ici il n'est plus question de bébés, mais de sirènes. Prenons une population de sirènes, dans leur grotte sous-marine, se nourrissant d'une algue délicieuse. En l'absence de lumière, les sirènes sont aveugles, car elles n'ont nul besoin de voir pour subsister. Supposons que leur grotte s'effondre, et qu'elles doivent sortir pour trouver leur nourriture. C'est là qu'interviennent les ennemis : des aigles qui leur plongent dessus pour les manger. Or, dépourvues de vue, les sirènes sont démunies face à cette menace venue du ciel.

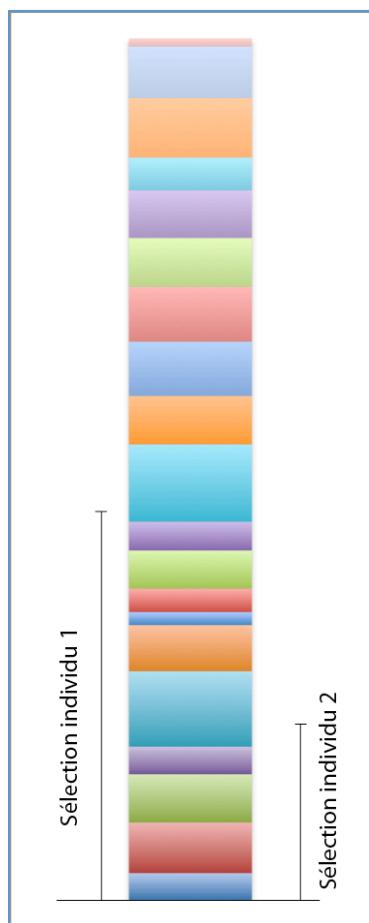
Mais un jour, une portée de sirène naît et elle est un peu particulière, la peau de son front est sensible à la lumière, et donc, ces sirènes survivent plus longtemps que les autres car elles savent éviter les aigles en replongeant lorsque le ciel s'obscurcit.

Comme elles survivent plus longtemps, leurs descendants sont plus nombreux, et certains héritent de leur mutation génétique. Au fil du temps, la population évolue : une sirène plus grande par exemple, apportera un autre avantage car elle pourra se reproduire plus efficacement, en appelant de plus loin les partenaires potentiels. Un beau jour, les deux types de sirènes finissent par se rencontrer, engendrant un nouveau type de sirènes, plus grandes et dotées de vision, donc plus adaptées au problème qui leur est donné : survivre.

À priori, tout cela semble sans intérêt pour le domaine informatique, mais cela nous permet de comprendre le fonctionnement de l'évolution et donc de l'algorithme basé sur cette théorie.

L'algorithme ne va donc pas trouver une solution à un problème posé, ni générer un individu optimal. Mais il va donner lieu à une population d'individus plus aptes en moyenne que les générations précédentes.

On commence donc par une population, un certain nombre de *chromosomes* (nos sirènes) contenant chacun une liste de gènes qui codent chacun un trait de caractère des individus (degré de vision, taille, ...). Cette population voit ensuite chacun de ses membres testé pour sa performance face au système. Une cote de performance est ensuite attribuée à chaque individu.



Principe de la Roulette wheel

Une fois tous les membres testés, on applique un algorithme de choix, appelé roulette wheel pour sélectionner des individus de la population afin de les faire se reproduire. La roulette wheel consiste à accorder à chaque individu de la population une chance d'être choisi proportionnelle à son propre score de performance. Concrètement, on commence par sommer toutes les performances, puis, on tire un nombre au hasard entre 0 et la somme obtenue. On parcourt ensuite les individus – toujours dans le même ordre !! – en s'arrêtant quand on a dépassé le nombre aléatoire trouvé : l'individu choisi est alors le dernier individu dont on a ajouté la performance.

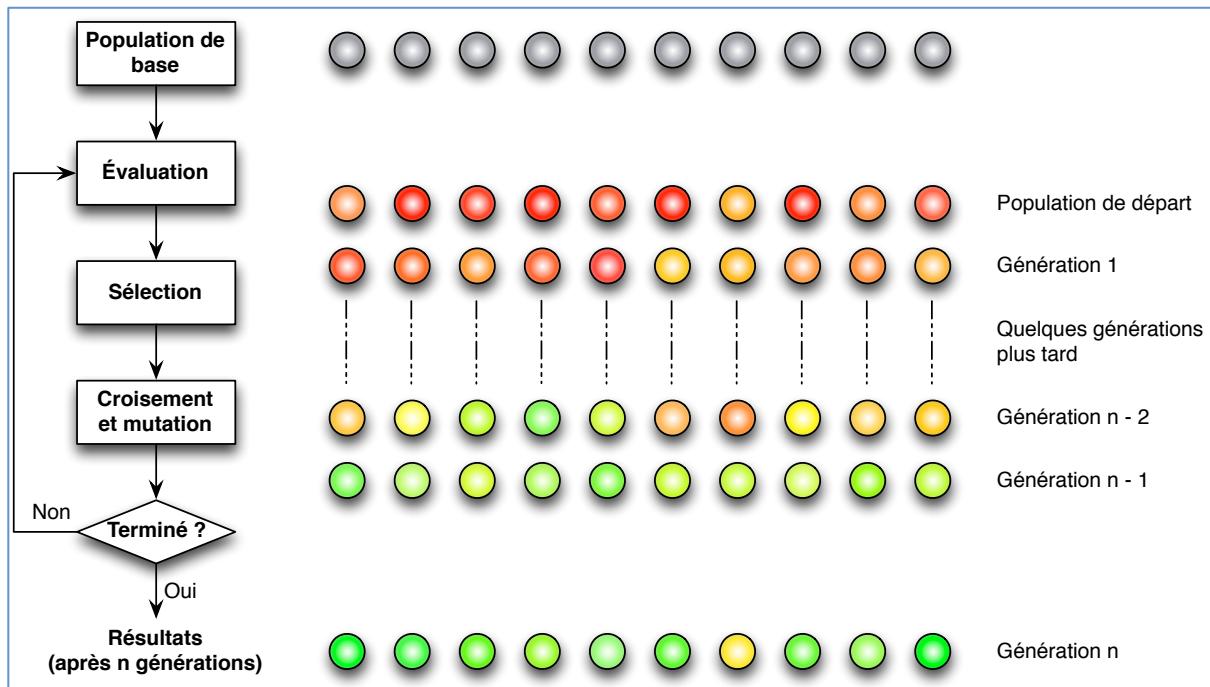
Cette méthode nous assure que les individus les plus choisis seront ceux dont la performance était plus importante, donc nos sirènes qui voient, survivant plus longtemps, auront une meilleure performance et seront plus choisies pour passer leur gène bénéfique aux générations suivantes.

Une fois que deux individus ont été choisis, il faut les croiser pour obtenir deux individus reprenant les gènes des parents. Dans le cas qui nous occupe, les deux individus résultant de l'opération disposent de gènes appartenant aux deux parents, afin que si une grande sirène s'accouple avec une sirène qui voit, les enfants puissent avoir les deux avantages pour eux.

Pour favoriser davantage les individus les plus aptes, on applique une technique nommée élitisme et qui consiste à garder, d'une génération sur l'autre, le ou les quelques meilleurs individus de la génération précédente. Cela permet de favoriser encore ces individus et donc les gènes portés par ceux-ci. On favorise donc la convergence vers une population plus apte.

Enfin, après chaque croisement d'individus, il y a une chance que les nouveaux individus aient mutés (apparition de la vision, taille plus importante, ...) et donc qu'un ou plusieurs de leurs gènes aient muté pour développer une nouvelle aptitude, caractéristique qui, peut-être, se révèlera cruciale pour l'évolution de l'espèce (ou peut être que ce sera une tare et que l'individu ne se reproduira donc plus du tout, ou très peu, limitant ainsi son influence sur l'évolution de l'espèce).

Au final, pour peu que la population ait une taille suffisamment importante (une population de 5 individus ne permettra pas un apprentissage ou une optimisation de l'espèce), les individus vont devenir de plus en plus aptes à résoudre le problème donné, et finiront par atteindre un point de quasi perfection dans sa résolution. En théorie, cette optimisation est parfaite et une solution est trouvée à 100% dans le cas où la liberté est totale (= toute modification comportementale du chromosome est possible) et où le nombre de générations est infini. En pratique, on peut atteindre une solution au problème quasi optimale en un nombre raisonnable de générations.



III. Choix d'un algorithme : l'algorithme génétique

Étant donné le problème posé, à savoir apprendre à jouer à Ms. Pac-man le mieux possible via un apprentissage, il a très vite été clair pour nous que nous devions typer les différents problèmes auxquels chaque algorithme savait répondre le mieux, puis réaliser un parallèle avec le problème de Pac-man afin de faire notre choix. Nous avons donc associé chaque algorithme avec son/ses points forts face à des types de problèmes particuliers :

- Q-Learning : apprentissage non supervisé pour rechercher une solution à un problème, apprendre un comportement efficace face à ce problème
- Réseau de neurones : apprentissage supervisé efficace pour mémoriser une base de connaissance empirique et en retirer le meilleur
- Algorithme génétique : apprentissage non supervisé visant à développer des fonctionnalités dans le but de trouver un comportement optimal envers un système ou un problème donné

Il nous est assez vite apparu que le problème de l'apprentissage d'une bonne stratégie pour Ms. Pac-man se rapprochait le plus du type de problèmes résolus par l'algorithme génétique. En effet, nous recherchons un individu qui sera adapté pour survivre dans son environnement le mieux possible, telle une sirène devant se nourrir en évitant les aigles, notre Ms. Pac-man devra se nourrir en évitant les fantômes.

(J'ouvre une petite parenthèse ici, car notre choix d'implémenter l'apprentissage du comportement du Pac-man ne fut pas le premier. En effet, nous avons longtemps persisté dans des essais d'implémentation d'un apprentissage pour le comportement des fantômes, mais cela ne change en rien notre choix de type d'algorithme, excepté que l'on doit éviter le Pac-man quand on est mangeable et le manger sinon. Dans la suite de ce rapport, nous prendrons pour acquis que notre choix est celui de Pac-man)

C'est donc vers cet algorithme que nous nous sommes penchés dès le début. Ceci dit, d'autres facteurs nous ont influencés dans ce choix, tels que la nécessité de fournir des données de jeu au réseau de neurones pour qu'il apprenne, et le fait que le Q-Learning était choisi par de nombreux autres groupes. Mais surtout, ce qui nous a finalement décidé à choisir l'algorithme génétique était son incroyable capacité à nous apprendre des choses : en effet, nous ne connaissons pas la stratégie optimale pour gagner à Ms. Pac-man, pour obtenir des scores faramineux, et en implémentant un contrôleur basé sur l'algorithme génétique, tout pouvait arriver, notre population finale pouvait être du genre à chercher à manger le plus de pastilles, à survivre sans s'occuper des pastilles en fuyant sans cesse l'ennemi, à tourner en rond, ... puisque nous laissions libre cours à l'algorithme pour trouver un comportement.

IV. Adaptation de l'algorithme choisi au problème

Une fois l'algorithme génétique choisi définitivement, nous avons dû nous l'approprier et l'adapter au cas de Ms. Pac-man. Nous avons donc consulté de la documentation en ligne sur le sujet afin d'être certains de ne pas s'égarer dans une mauvaise direction et de bien comprendre le fonctionnement de cet algorithme pour le rendre le plus efficace et le plus clair possible.

Ensuite, la réelle difficulté fut de déterminer quels paramètres pourraient être contenus dans nos gènes. Difficulté augmentée par notre approche initiale de l'apprentissage des fantômes, sur lesquels nos choix de gènes n'ont que peu d'influence sur les performances, et donc sur lesquels l'apprentissage n'avait pas lieu. Une fois que nous sommes revenus sur ce choix malencontreux, nous avons pu avancer assez vite.

Nous avons donc implémenté nos gènes via un tableau de réels, compris entre 0 et 1 et exprimant chacun la propension de l'individu à un certain comportement (par exemple, la gourmandise : propension à vouloir manger tout ce qu'il peut, la couardise : propension à fuir les fantômes, etc...). Cette manière de faire a un avantage certain qui est sa simplicité pour ce qui est du croisement et des mutations : toutes sortes de techniques existent pour croiser deux tableaux, comme les techniques avec un ou des pivot(s) où des échanges entre les deux gènes se font avant/après/entre les pivots dans un sens, puis dans l'autre. Par exemple, si on prend un seul pivot, on considère deux chromosomes père et mère :

$p_1 \ p_2 \ p_3 \ p_4 \ p_5 \ p_6 \ p_7 \ p_8$
 $m_1 \ m_2 \ m_3 \ m_4 \ m_5 \ m_6 \ m_7 \ m_8$

On sélectionne un pivot aléatoire, mettons 6, et on inverse tous les gènes après le pivot :

$p_1 \ p_2 \ p_3 \ p_4 \ p_5 \ m_6 \ m_7 \ m_8$
 $m_1 \ m_2 \ m_3 \ m_4 \ m_5 \ p_6 \ p_7 \ p_8$

Une fois le croisement exécuté, la mutation est simple : une boucle sur chaque gène du chromosome muté et si la probabilité de mutation est atteinte pour un des gènes, on ajoute une variable normale centrée en 0 et d'écart-type 1, après quoi les valeurs obtenues sont bornées pour rester entre 0 et 1.

En plus de cela, une technique d'élitisme est appliquée et avant tout croisement, les deux individus ayant obtenu la meilleure performance sont réinjectés dans la nouvelle population. Cela augmente la vitesse de convergence de l'algorithme.

Enfin, la méthode de sélection des membres est celle de la roulette wheel expliquée ci-dessus et implémentée de manière standard via :

- une première boucle qui somme les performances
- le calcul d'un nombre aléatoire entre 0 et le total obtenu
- une boucle pour retrouver le gène correspondant à ce nombre dans la population.

Pour des raisons de temps de calcul, lié au nombre de parties à simuler pour tester chaque individu nous avons choisi une population relativement petite : 100 individus, et nous avons choisi de les tester chacun 3 fois pour que la performance soit une moyenne, ce qui lisse les mauvaises performances dues à un hasard malencontreux qui peut se produire.

Cette performance, nous avons choisi deux manières de la calculer :

$$\text{Performance} = \text{score obtenu} (\text{moyenne des 3 parties})$$

$$\text{Performance} = (\text{score obtenu} (\text{moyenne des 3 parties}))^2$$

Nous avons assez vite constaté une convergence plus rapide et meilleure avec la seconde solution, comme vous le verrez dans la partie « Analyse des résultats » plus loin dans ce rapport.

La dernière étape a donc été de créer un contrôleur pour Ms. Pac-man prenant en paramètre un tableau de gènes et effectuant les jets de hasards nécessaires pour déterminer la direction à suivre.

Il nous a fallu un moment pour cerner code du jeu et pour en comprendre le comportement lors de demandes de localisation d'éléments absents, ou introuvables. Dès lors, nous avons pu empêcher les erreurs et effectuer les contrôles manuels nécessaires. Ce n'est qu'alors que nous avons pu nous lancer dans la création des différents gènes et de leur effet sur la décision de la direction à suivre.

Pour pouvoir prendre en compte chaque gène et non pas juste le premier, il nous est vite apparu que la décision ne devait être prise qu'en fin de fonction. Il nous fallait un système pour tenir compte de l'influence qu'aurait chaque gène. Nous avons trouvé la solution dans un tableau de 4 cases représentant la propension de Pac-man à se déplacer suivant les 4 directions : haut, droite, bas et gauche. Nous avons donc naturellement choisi 4 gènes qui représentent chacun la propension de base à se déplacer dans telle ou telle direction et initialisé un tableau avec ces 4 valeurs. Ensuite, pour chaque gène choisi, en fonction du jet de hasard, et de la décision qui en découle, on augmente la propension à se diriger dans la direction choisie. Si le gène demande une réaction inversément proportionnelle à la distance qui sépare Ms. Pac-man d'un élément (pastille ou fantôme), c'est une valeur de un sur cette distance qui est ajoutée à la direction voulue.

Une fois tous les gènes parcourus, on obtient un tableau résumant la propension à se diriger dans chaque direction et il nous suffit de choisir la direction pour laquelle la propension est la plus forte.

Les gènes que nous avons choisis sont détaillés dans le code lui-même mais en voici un descriptif concis :

PROPENSION_HAUT : propension qu'a l'individu à aller vers le haut

PROPENSION_DROITE : propension qu'a l'individu à aller vers la droite

PROPENSION_BAS : propension qu'a l'individu à aller vers le bas

PROPENSION_GAUCHE : propension qu'a l'individu à aller vers la gauche

GOURMANDISE : propension qu'a l'individu à chercher à manger ce qu'il peut et qui est proche de lui

COUARDISE : propension qu'a l'individu à fuir ses ennemis

AGRESSIVITE : propension qu'a l'individu à attaquer les ennemis faibles

STUPIDITE : propension qu'a l'individu à aller dans une direction aléatoire

MAL_DE_VIVRE : propension qu'a l'individu à se suicider en fonçant vers l'ennemi le plus proche

INDECISION : propension qu'a l'individu à faire demi-tour

DIST_PEUR : distance à partir de laquelle l'individu commence à avoir peur de ses ennemis

DIST_GOURMAND : distance à partir de laquelle l'individu commence à avoir envie de manger ce qu'il voit

DIST_AGRESSION : distance à partir de laquelle l'individu commence à avoir envie d'attaquer les ennemis

Nous avons effectué nos tests non seulement avec les deux modes d'évaluation de la performance, mais aussi en tenant compte ou non des distances (les 3 derniers gènes définis).

V. Mode d'emploi pour l'intégration au projet Ms. Pac-man

Le déploiement de notre algorithme et son intégration au projet est simple, il suffit d'ajouter les deux classes fournies (« NotreAlgorithmeGenetique » et « PacmanControllerApprenant ») au projet Ms. Pac-man VS ghosts, respectivement dans les packages algorithme_apprentissage et game.controllers, et de modifier la méthode runExperiment de la classe Exec du projet pour qu'elle renvoie un float, et lui faire renvoyer la variable du score moyen (avgScore/trials).

Ceci fait, il suffit de lancer la méthode main de la classe «NotreAlgorithmeGenetique».

Pour choisir la performance, il suffit de changer le booléen CARRE, avec la valeur true pour prendre comme performance le score², false pour prendre le score.

VI. Analyse des résultats

Sur base des tests effectués, nous avons pu constater que l'implémentation de notre algorithme génétique était fonctionnelle, et que notre population de départ apprenait bien une stratégie de jeu, multipliant, en 100 générations, son score initial par 25 contre des fantômes de tests aléatoires. Cela a été mis en évidence par un tableau mettant en forme les données brutes récoltées.

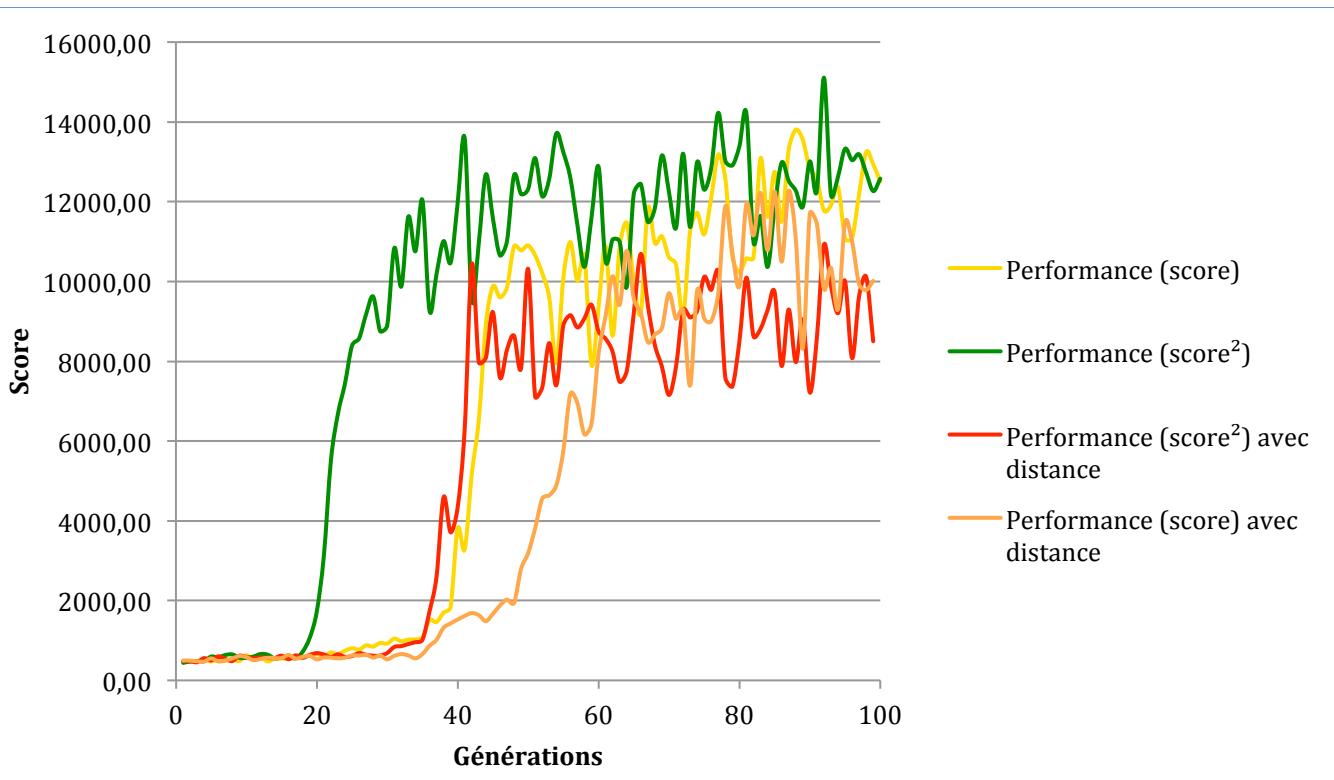
Voici donc ces résultats, colorés du rouge au vert en fonction de la performance :

Generation	Sans notion de distance		Avec notion de distance	
	Performance (score)	Performance (score ²)	Performance (score) avec distance	Performance (score ²) avec distance
1	457,97	438,63333	503,5667	485,33337
2	465,20	472,70004	496,06668	481,09998
3	485,00	483,96664	479,79996	454,63336
4	537,37	481,00003	470,86667	557,3667
5	476,87	598,20013	551,8999	500,10004
6	538,50	578,2333	476,0667	602,4001
7	488,80	635,73346	498,80008	539,53345
8	522,40	650,49994	548,03326	484,93332
9	484,20	564,49994	609,69995	617,2666
10	626,67	566,09985	581,89996	588,93335
11	539,50	596,7999	505,1334	551,79987
12	589,13	660,2333	536,10004	578,63336
13	476,60	645,3	567,6001	560,56665
14	544,30	553,23334	558,4	559,8334
15	547,43	567,76666	558,66675	619,66675
16	539,73	586,4334	637,8334	532,23334
17	560,87	555,9332	546,3999	626,9667
18	563,60	720,7	588,86676	567,7999
19	637,10	1069,6665	618,16675	637,9
20	656,67	1727,3995	527,80005	683,50006
21	604,23	3118,2334	574,1333	648,4333
22	701,63	5504,966	570,1001	586,7666
23	656,80	6682,5996	553,4666	652,5001
24	743,90	7440,765	569,0999	588,9666
25	805,80	8392,333	625,0666	610,0333
26	770,53	8571,231	623,2333	684,2
27	878,13	9177,835	641,86664	644,0668
28	846,27	9622,935	571,0001	617,73334
29	937,73	8752,3	619,4332	622,1334
30	921,63	8899,367	528,1333	687,49994
31	1048,13	10837,869	614,06665	837,2667

32	980,67	9874,767	659	859,1665
33	1023,93	11626,335	625,90015	912,03326
34	1028,37	10757,164	551,9334	957,53314
35	1078,73	12036,398	659,6666	1018,7
36	1508,90	9270,766	866,8334	1726,8666
37	1463,00	10189,333	1018,3	2598,6665
38	1703,03	11012,099	1318,2334	4592,5664
39	1838,37	10472,466	1428,0665	3711,4993
40	3806,93	11968,865	1525,7997	4337,5664
41	3285,67	13589,534	1617,1669	6310,0996
42	5169,70	9538,833	1687,2332	10435,199
43	6524,30	10967,17	1635,2332	7975,766
44	8930,40	12685,668	1484,6332	8085,4346
45	9878,30	11585,063	1665,6996	9234,867
46	9601,44	10656,1	1877,0668	7593,9004
47	9850,50	11009,134	2024,7	8286,532
48	10880,10	12660,001	1939,6	8641,165
49	10777,13	12194,268	2792,7336	7826,8667
50	10902,67	12312,866	3186,3665	10318,534
51	10653,73	13095,197	3813,0662	7121,267
52	10211,90	12137,764	4554,001	7337,567
53	9599,47	12572,135	4640,3677	8455,935
54	7955,93	13710,968	4885,8677	7397,2344
55	10080,63	13240,197	5750,9336	8894,667
56	10989,77	12599,799	7179,3994	9156,099
57	10019,13	11419,768	6949,2676	8848,267
58	10636,20	10366,965	6165,0664	9075,5
59	7905,63	11597,799	6428,768	9418,099
60	9343,70	12880,832	8174,2676	8750,333
61	10892,40	10505,301	9131,966	8566,166
62	8636,47	11052,834	10132,169	8236,166
63	10883,07	10997,732	9416,133	7486,034
64	11471,33	9862,2	10770,934	7750,201
65	10017,47	12194,867	9571,899	9191,9
66	9226,70	12445,599	9208,231	10691,701
67	11831,80	11498,265	8482,833	9408,866
68	10961,66	11839,732	8675,165	8406,134
69	11134,20	13160,701	8830,699	7866,6646
70	10600,33	12245,165	9705,4	7155,1655
71	10428,93	11341,2	9067,934	7864,43
72	9129,94	13205,3	9284,1	9287,901
73	11258,43	11364,735	7388,768	9099,001
74	11724,10	12995,701	9772,836	9262,764
75	11182,34	12300,102	9077,401	10115,102
76	12096,90	12845,232	9005,934	9790,268

77	13187,60	14231,429	9704,169	10246,935
78	12584,73	13025,433	11881,968	7595,3677
79	10625,87	12909,866	10731,268	7366,0654
80	10223,80	13386,603	9883,268	8495,034
81	10583,17	14210,969	11947,7	10096,701
82	10589,53	10989,534	11157,397	8630,432
83	13090,14	11639,966	12220,867	8813,2
84	11617,50	10363,166	10783,669	9277,698
85	12749,97	11843,202	12259,937	9735,633
86	11497,60	12980,366	10497,032	7883,265
87	13306,80	12524,534	12272,567	9298,199
88	13806,77	12280,799	11084,199	7979,2007
89	13553,90	11871,732	8317,137	9046,632
90	12820,86	13010,801	11707,3	7215,4336
91	12569,80	12272,331	11449,3	8588,935
92	11782,44	15111,298	9814,767	10917,567
93	11912,27	12171,032	10339,799	9891,9
94	12370,50	12627,098	9303,635	9204,766
95	11082,47	13326,63	11477,798	10008,733
96	11085,57	13041,666	11001,001	8077,935
97	12190,87	13188,395	9927,7	9639,1
98	13242,11	12727,435	9787,7	10116,399
99	12937,83	12267,269	10015,003	8502,102
100	12509,67	12576,965	11837,769	8555,531

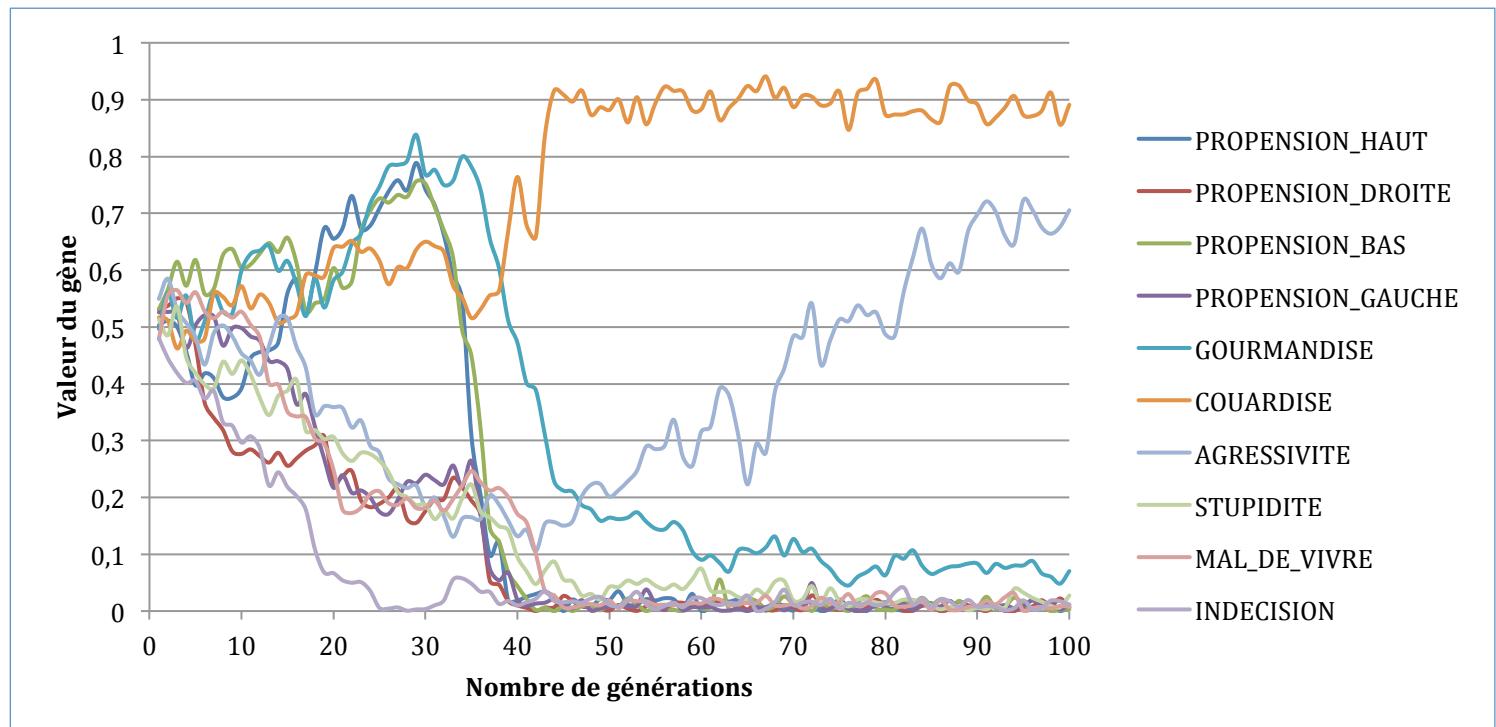
On constate aisément en regardant ce tableau que la progression est maximale si on ne tient pas compte de la distance et si on prend pour mesure de la performance le carré du score. Ceci est encore plus visible sur le graphique correspondant :



Ces résultats mettent donc en avant la supériorité du calcul de la performance sur bas du score au carré, que l'on prenne en compte la distance ou pas. Ceci s'explique facilement par le fait que la mise au carré du score amplifie d'autant plus les écart entre les bons éléments et les mauvais, conférant aux meilleurs des chances d'autant plus grandes d'être choisis pour les croisements.

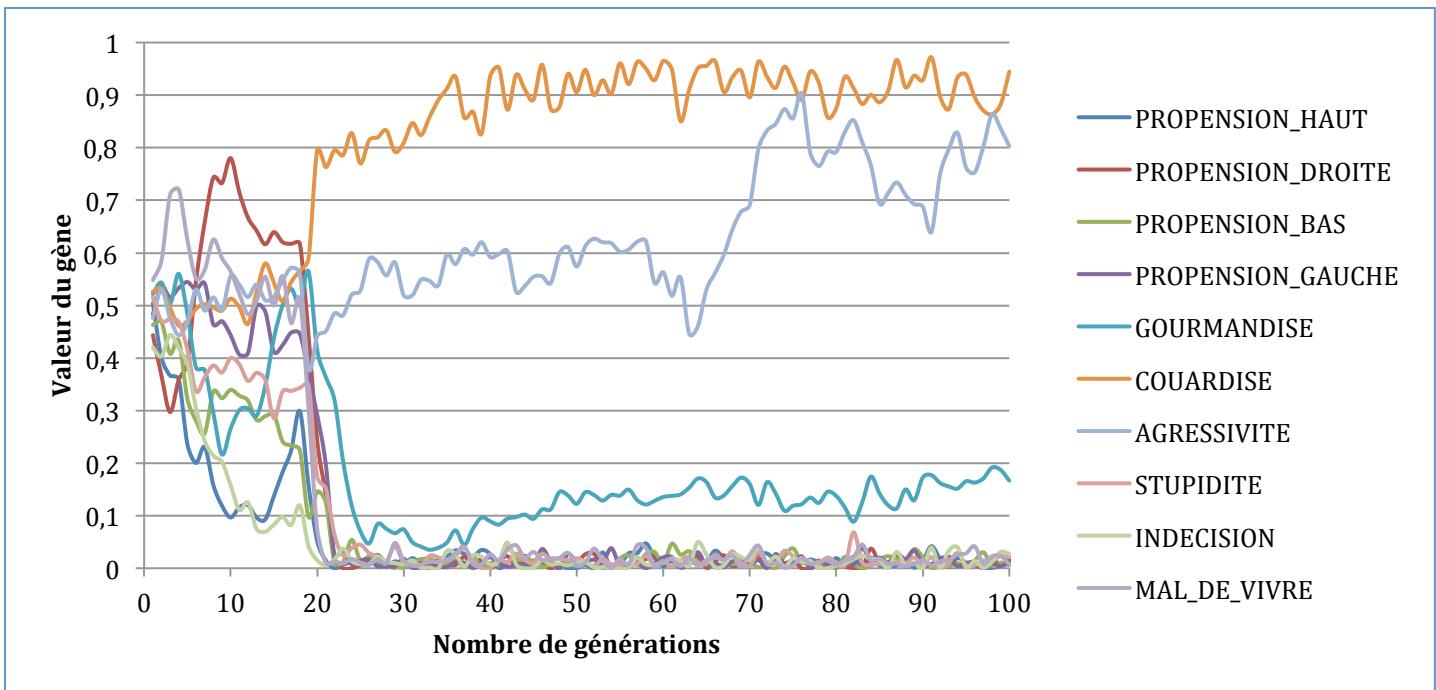
Outre ces données sur les scores, qui permettent de visualiser l'évolution de l'apprentissage au fil des générations, nous avons réalisé des graphiques reprenant l'évolution des différents gènes dans les différents cas :

Performance (score) :



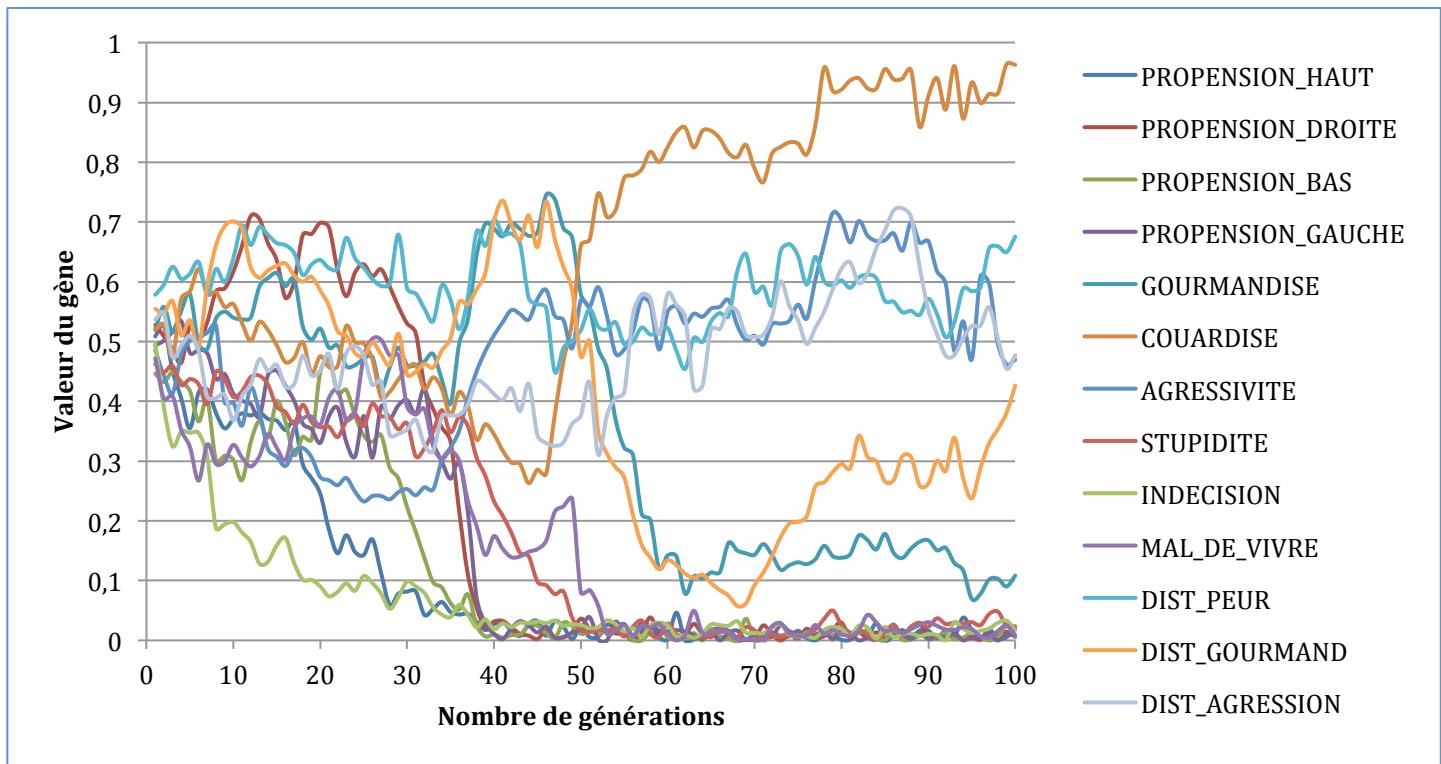
Ce graphique nous montre que les caractéristiques les plus utiles sont la couardise, l'agressivité et, dans une moindre mesure, la gourmandise. Les autres variables se stabilisent très vite autour de zéro.

Performance (score²) :



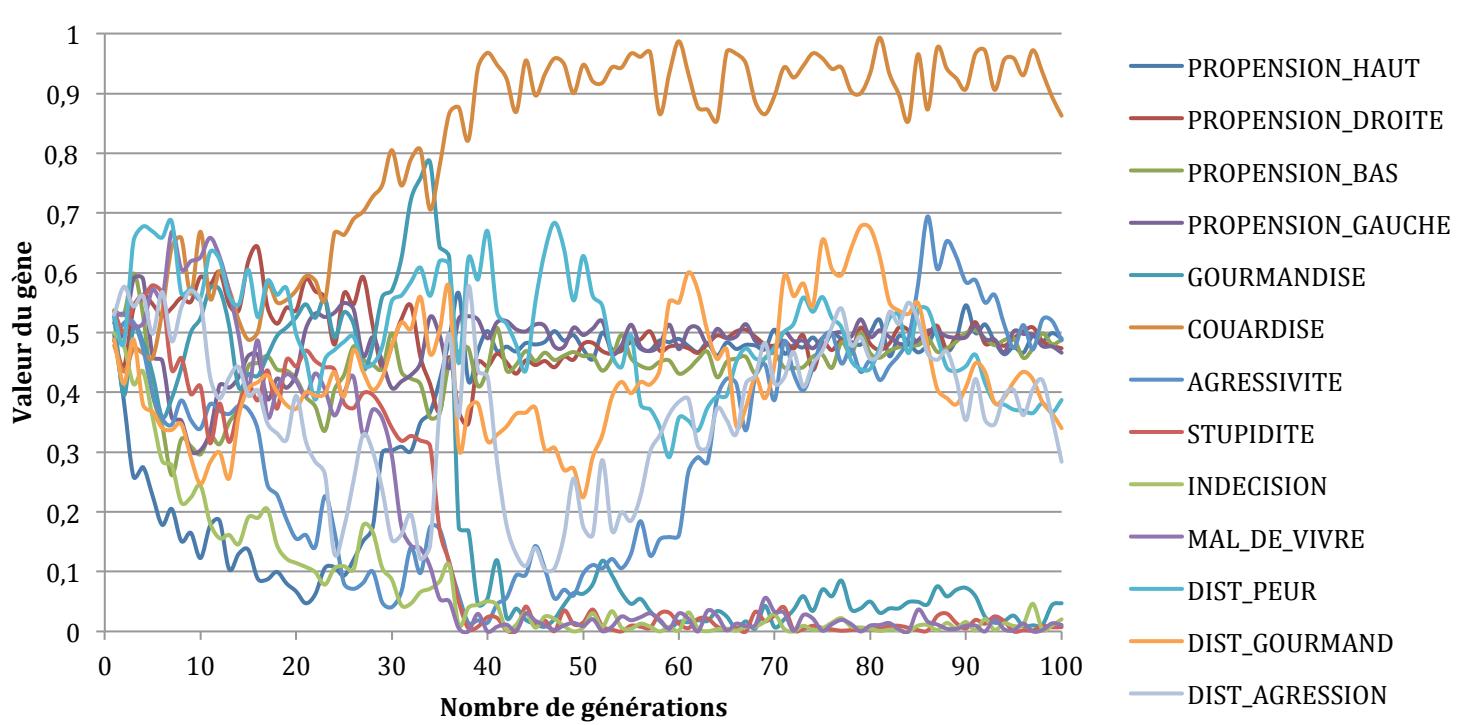
Nous observons ici des résultats similaires au graphique précédent, quoique plus rapides à se mettre en place. Ceci est dû, comme expliqué précédemment, au choix du score au carré comme mesure de performance qui accélère la convergence.

Performance (score) avec distance :



Ce graphique-ci nous expose un comportement différent, dû à la présence des paramètres de distances dans les gènes. On y constate une évolution parallèle de la distance d'agression et de la propension à l'agressivité. On y voit également un parallèle entre la distance de peur et la couardise, bien que la distance tende à rester plus basse que la couardise à laquelle elle est liée (nous rappelons que les distances sont exprimées en % de la distance maximale dans un labyrinthe). La distance à laquelle le Pac-man devient gourmand semble croître vers la fin, mais reste néanmoins inférieure aux autres distances.

Performance (score²) avec distance :



Enfin, nous observons sur ce graphique une stabilisation des distances légèrement inférieure à la moitié de la distance maximale dans le labyrinthe. Les autres paramètres semblent avoir pris des valeurs similaires aux cas précédents : couradise largement en tête, aggressivité un peu plus basse et gourmandise en queue de peloton, proche de toutes les autres variables. La grosse différence se situe au niveau des valeurs importantes prises par les propensions aux 4 directions : cela s'explique simplement par le fait que l'on constate que leur influence est nulle, tant que les 4 valeurs ont plus ou moins la même valeur, aucune direction n'est donc à priori favorisée.

Comme on peut donc le constater, ces graphiques sont assez semblables. Il en ressort une stratégie gagnante globale : la couardise. En effet, ce gène est celui dont la croissance est la plus forte au terme des générations et ce dans les 4 cas testés. Notre but en choisissant l'algorithme génétique, à savoir lui permettre de trouver un comportement optimal, est atteint. Nous apprenons donc que le comportement vainqueur est la fuite, et l'attente à l'abri de la fin du niveau. Ms. Pac-man récolte ainsi les points des pastilles restantes.

Cela reste vrai contre des fantômes aléatoires et compte tenu des possibilités limitées de notre pac-man en terme de comportements (tous les comportements ne pourront pas être développés, par exemple notre pac-man ne développera jamais, de par la structure de son contrôleur et les gènes choisis, de comportement intelligent prenant en compte les positions des 4 fantômes, ...). En exécutant les tests avec un autre contrôleur pour les fantômes, un autre comportement pourrait émerger, optimisé pour lutter contre ce nouveau problème.

VII. Conclusion

Ce projet, bien que flou dans notre esprit au début, nous a permis de mettre en pratique et de nous approprier le fonctionnement des concepts vus en cours théorique. Il a constitué un challenge, nous obligeant à penser la programmation autrement, en perdant le contrôle sur les décisions prises et en laissant la main à notre création. La rédaction de ce rapport nous a également poussé à nous pencher sur les algorithmes proposés, et à nous renseigner sur un algorithme nouveau (le Q-Learning).

Sur le plan de l'intégration au projet Ms. Pac-man, nous avons dû apprendre à lire, à comprendre et à nous approprier des sources complexes et peu commentées, afin d'en tirer un maximum d'informations. Nous avons donc pu, au travers d'une des premières API que nous rencontrions, nous frotter au développement de modules tournant au-dessus d'un moteur tiers pour y apporter une fonctionnalité (ici, un pac-man qui apprend).

Enfin, en nous forçant à mettre en place des outils caractéristiques d'une partie très ésotérique de la programmation, ce projet nous aura poussé à rechercher et à traduire les concepts nouveaux et parfois très mathématiques en des termes concrets et parlants. Nous avons ensuite dû modéliser les paramètres plus instinctifs du jeu dans une structure et une forme que l'algorithme pouvait digérer, faisant ainsi le lien entre le monde des humains, dans lequel ces règles (ne pas se faire manger c'est mieux, avoir un bon score c'est mieux, mourir c'est mal, ...) sont établies tacitement, et un monde numérique où un nombre ou une variable n'ont que le sens qu'on leur donne, où l'instinct est inexistant et où seule la logique règne.

En conclusion, nous sortons de ce projet grandis et plus expérimentés mais aussi contents d'avoir mené à bien une tâche d'une telle complexité. En espérant que notre travail vous aura plu au regard de l'énergie et de l'enthousiasme que nous y avons mis.

Merci de votre lecture !

VIII. Bibliographie

Intelligence Artificielle - l'algorithme du Q-Learning

http://thierry.masson.free.fr/IA/fr/qlearning_apropos.htm, Thierry Masson, dernière édition : juin 2003

Q-Learning – Kranf site

<http://www.applied-mathematics.net/qlearning/qlearning.html>, Dr. Ir. Frank Vanden Berghen

Q-learning - Wikipédia

<http://en.wikipedia.org/wiki/Q-learning>, dernière édition : aout 2011

Les réseaux de neurones

<http://igm.univ-mlv.fr/~dr/XPOSE2002/Neurones/>, Rachid Ladjadj, dernière édition : 2003

Introduction aux Réseaux de Neurones

<http://www.sylborth.com/nn.php>, Sylvain Barthelemy, dernière édition : juin 2000

Artificial neural network – Wikipédia

http://en.wikipedia.org/wiki/Artificial_neural_network, dernière édition : décembre 2011

Neural network - Wikipédia

http://en.wikipedia.org/wiki/Neural_network, dernière édition : novembre 2011

Réseau de neurones artificiels - Wikipédia

http://fr.wikipedia.org/wiki/Réseau_de_neurones_artificiels, dernière édition : décembre 2011

Fonction gaussienne - Wikipédia

http://fr.wikipedia.org/wiki/Fonction_gaussienne, dernière édition : décembre 2011

Genetic Algorithms in Plain English – AI Junkie

<http://www.ai-junkie.com/ga/intro/gat1.html>, dernière édition : janvier 2005

Introduction to Genetic Algorithms

<http://www.rennard.org/alife/english/gavintrgb.html>, Jean-Philippe Rennard, dernière édition : mai 2006

Algorithme génétique - Wikipédia

http://fr.wikipedia.org/wiki/Algorithme_g%C3%A9n%C3%A9tique, dernière édition : octobre 2011

Rétropropagation du gradient - Wikipédia

<http://fr.wikipedia.org/wiki/Rétropropagation>, basé sur le livre : Patrick van der Smagt, An Introduction to Neural Networks, 1996