Bordeaux INP
ENSC

PROJET INFORMATIQUE

Algorithmes Génétiques et Réseaux Neuronaux

Étudiant	Corentin Bienassis

Projet	2A/INFO
Tuteur	Mr. FAVIER
Date de début	26/01/2017

Table des matières

Introduction

Problématique

Le projet consiste à appliquer de manière concrète les notions de réseaux neuronaux et d'algorithmes génétiques en codant une Intelligence Artificielle (appelée IA par la suite) pour un jeu basique (que nous coderont également). Ce projet s'inscrit dans une démarche d'apprentissage de la notion d'apprentissage automatique, en se penchant plus précisément sur la branche des réseaux neuronaux et des algorithmes génétiques, de plus en plus en vogue (bien que la théorie date des années 80) et dont l'accessibilité à un public étudiant a été rendu possible avec l'avènement d'Internet.

La naissance de l'idée

Avant même d'avoir des cours consacrés aux réseaux neuronaux et aux algorithmes génétiques dans le cadre du cursus de l'ENSC, ces derniers sujets m'intéressaient fortement. Les champs d'applications de ces notions sont très vastes et donnent des résultats très impressionnants.

Sur YouTube, et Internet en général, de nombreuses vidéos consacrées à ces sujets fourmillent. Que ce soit l'exposition de résultats concrets obtenus, des vulgarisations de ces concepts ou encore des conférences TEDx relatives à des problématiques adjacentes, l'ensemble des informations nécessaires étaient présentes pour pouvoir se lancer dans le bain de ces concepts plutôt abstraits.

Préexistant

Le jeu

Le premier élément existant est le jeu en lui-même. Accessible depuis l'interface de Google Chrome et implémenté en Javascript, l'ensemble du code est extractible. Cependant, comme nous allons le voir par la suite, j'ai recodé le jeu de manière à pouvoir y intégrer par la suite mon code personnel. Quoiqu'il en soit, les sprites (images utilisées dans le jeu), les sons, et d'autres éléments ont pu être repris par lors de ce projet.

Des exemples existants

L'idée d'appliquer les notions d'algorithmes génétiques et de réseaux neuronaux au jeu de Google Chrome ne vient pas de moi : d'autres programmeurs ont eu la même ambition dans le passé. Parmi eux, on retrouve Tony Ngan, qui semble avoir eu l'idée originelle puis, surtout, Ivan Seidel, un brésilien ayant fait une vidéo de vulgarisation très accessible sur ce sujet précis.

Le jeu

Règles

Comme expliqué précédemment, le jeu est issu de Google Chrome. Il s'agit d'un *Easter Egg*, c'est-à-dire d'un jeu « caché » dans le navigateur. Pour y accéder, il faut déjà n'être connecté à aucun réseau (wifi ou Ethernet). Une page s'affiche alors affichant un message d'erreur ainsi qu'un T-Rex. Pour déclencher le jeu, il suffit alors d'appuyer sur la barre espace.

Le jeu a initialement été développé pour le navigateur Chromium, la version Linux de Google Chrome.

Le but est simple : le joueur appuie sur la flèche du haut pour faire sauter le T-Rex et sur la flèche du bas pour le faire se baisser. Une fois lancé, le T-Rex court et ne s'arrête que lorsqu'il percute un obstacle. Le but du jeu est d'atteindre le meilleur score possible, sachant que la vitesse du jeu augmente petit à petit au fur et à mesure du temps. Un aperçu du jeu est disponible via ce lien.

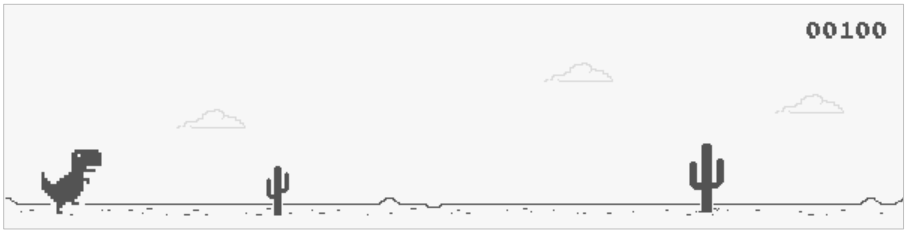


Figure 1 - Aperçu du jeu

Un premier problème

Bien que le code du jeu déjà existant soit extractible, il s'agit de plus de 2000 lignes de code peu compréhensibles et ne concernant pas uniquement l'algorithme du jeu mais également sa portabilité sur différentes plateformes. L'idée de recoder le jeu s'est donc imposée assez rapidement.

Dans un premier temps, recoder le jeu permet de ne pas avoir de problème quant à la compréhension du code déjà existant. Dans un second temps, recoder le jeu permettra par la suite d'avoir une totale compréhension de chaque partie du code et donc de pouvoir l'intégrer facilement à mon algorithme génétique. Enfin, cela nous permet d'effectuer quelques modifications pertinentes pour la suite, comme par exemple intégrer la possibilité de faire jouer plusieurs dinosaures dans une même partie.

Architecture logicielle

Diagramme de classe

Le code est organisé en une page HTML, une page CSS et un script de lancement en Javascript (lien sur un cours rédigé par Baptiste Pesquet) auxquels s'ajoutent neuf autres classes en Javascript comme présentées dans le diagramme de séparation du code ci-dessous :

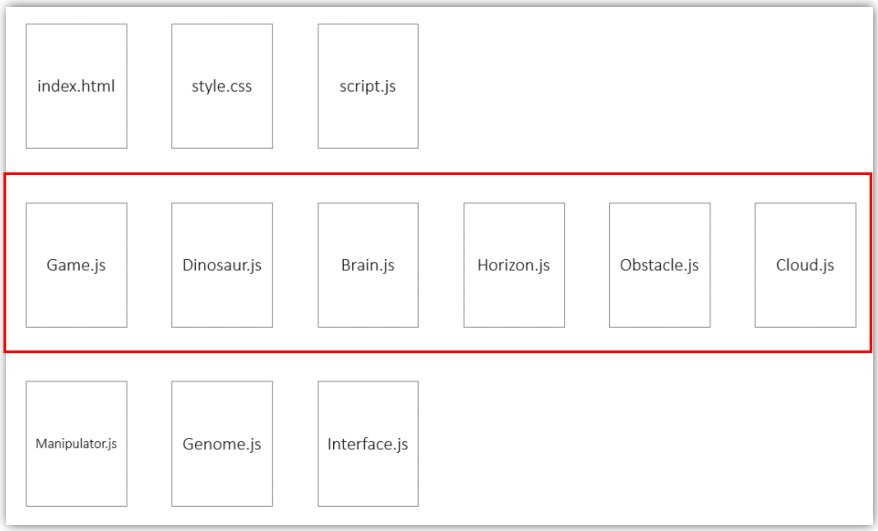


Figure 2 - Séparation du code

Les classes correspondant directement au jeu en lui-même sont encadrées en rouge. Ne nous préoccupons pas des autres classes pour le moment.

Arborescence des fichiers

Chaque fichier du projet est rangé au sein d'une arborescence soignée permettant de le retrouver facilement.

Librairie utilisée

La librairie utilisée pour l'implémentation du code est P5.js, une surcouche du langage Processing (un peu plus connu), ce dernier étant lui-même une surcouche de Javascript. La librairie P5.js permet d'avoir un code préparé avec une boucle de jeu rafraîchissant l'écran à une fréquence de 60 images par secondes. Elle possède également un ensemble de fonctions très pratiques pour afficher les sprites ou encore de nombreuses fonctions mathématiques bien pratiques. Cependant, cette librairie a posé des problèmes par la suite comme cela sera expliqué dans la partie *Difficultés Rencontrées*.

Réflexion basique

Pour rappel, le but du projet est de faire une IA pour que le dinosaure fasse le meilleur score possible. La première approche, plutôt évidente, est de coder un algorithme basique mais fonctionnel ressemblant au modèle suivant :

```
if (condition)
    jump();
if (condition)
    duck();
```

Figure 3 - Prototype d'IA basique

Cependant, ce genre d'algorithme n'apprend pas par lui-même, et ne peut pas s'adapter à son environnement si l'on change les paramètres du jeu. C'est pourquoi nous allons utiliser un réseau neuronal.

Réseaux neuronaux

Approche générale

La théorie des réseaux neuronaux est née dans les année 80, mais a été mise de côté au profit d'autres branches de l'IA car les systèmes n'étaient pas encore assez performants à l'époque pour exploiter la théorie de manière pertinente. Il s'agit d'une théorie mathématique bio-inspirée, qui mime le fonctionnement des neurones du cerveau biologique tel que nous le connaissons.

Concrètement, un réseau neuronal, aussi appelé perceptron, est un réseau constitué (dans l'immense majorité des cas) de couches successives de neurones, eux même reliés entre eux par des connexions.

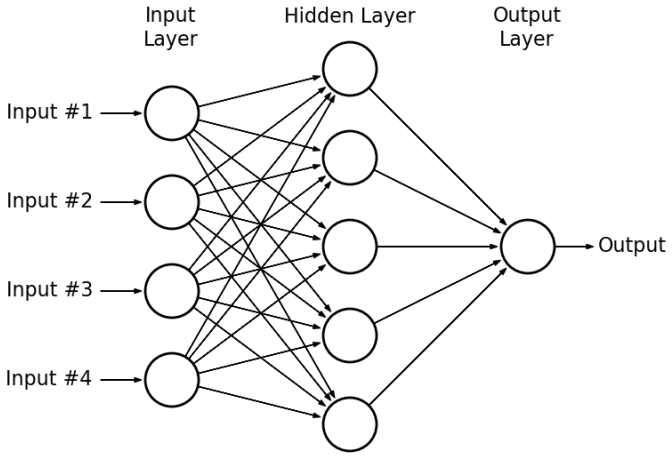


Figure 4 - Organisation d'un Réseau Neuronal (ou perceptron) simpliste

La première couche (*layer*, en anglais) est appelée la couche input (*input layer*). Elle est unique et ce, peu importe le nombre de couches totales du perceptron. La dernière couche est appelée couche output (*ouput layer*). Elle est également unique. Les couches du milieu sont appelées couches cachées (*hidden layers*). Leur nombre varie selon la complexité du problème à résoudre et est choisi par le créateur. Le nombre de neurones sur chaque couche peut être différent. Il est aussi choisi par le créateur du perceptron.

Application au projet

Une librairie bien pratique

Le projet étant réalisé en javascript, une librairie nommée synaptic.js s'est vite avérée indispensable.

Il aurait été possible de recoder soi-même les concepts de perceptron, neurone et connexion, mais lors du choix, l'objectif principal était d'arriver à un résultat fonctionnel en priorité (un T-Rex qui apprend). De plus, avec l'utilisation d'une librairie, on s'assurait d'ores et déjà d'un code fonctionnel, testé et maintenu. Enfin, lors d'un travail en entreprise, il est courant d'utiliser des librairies existantes afin d'améliorer la vitesse de production ainsi que la fiabilité du produit (cf. cours de l'entreprise Quorum). Il était donc pertinent d'apprendre à utiliser un travail existant sans avoir à réinventer la roue lors de ce projet.

Concrètement, ça donne quoi ?

Le perceptron associé à chaque T-Rex se base sur le modèle suivant. Chaque *Dinosaur* possède un *Brain* associé à un *Genome* lui-même possédant un *Perceptron*.

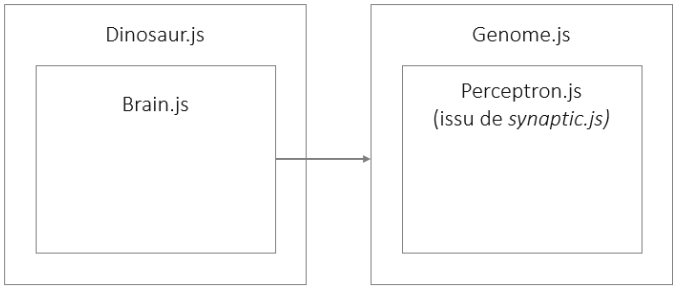


Figure 5 - Référence à Genome.js depuis Brain.js

Notre perceptron

Le choix du nombre de couches ainsi que du nombre de neurones par couche se fait plus ou moins selon le bon-vouloir du développeur et en fonction de la difficulté du problème à résoudre (quitte à changer la structure du perceptron par la suite pour augmenter la vitesse de convergence). Après, un chercheur expérimenté dans le domaine trouvera plus facilement, avec son savoir et son intuition, le meilleur paramétrage pour la création de son perceptron. Cependant, il faut déjà veiller à ce que le perceptron ne soit pas trop simpliste, auquel cas le T-Rex n'arrivera jamais à un comportement cohérent.

Le nombre de neurones d'entrée est, lui, choisi selon le nombre de données que l'on récupère via les capteurs.

Les capteurs

Pour utiliser le Perceptron, il nous faut dans un premier temps récupérer les données d'entrées, c'est-à-dire les données qui seront transmises à chaque neurone de la couche input. La récupération de ces données se fait depuis la classe Brain. Ici, on a besoin de différentes données, celles que l'on juge utiles pour que le T-Rex puisse apprendre. On choisit donc 4 « capteurs » :

1. La vitesse du jeu. Cette vitesse augmente à chaque obstacle franchi.
2. La distance séparant le T-Rex du prochain obstacle,
3. La largeur du prochain obstacle,
4. L'altitude du prochain obstacle si ce dernier est un ptérodactyle (0 si c'est un cactus).

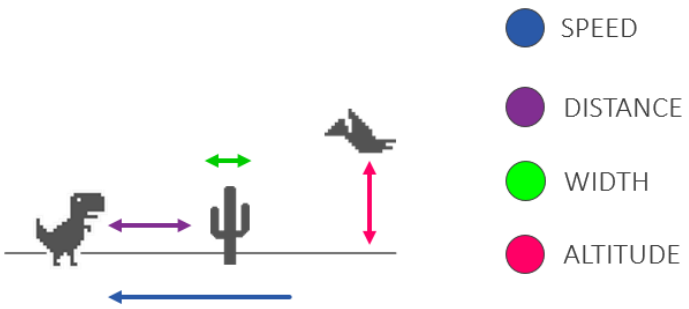


Figure 6 - Représentation des valeurs d'entrée (inputs)

Remarque : Dans le cas où l'on ne connait pas l'ensemble des paramètres nécessaires au T-Rex pour apprendre à jouer, il reste possible de relier chaque pixel de l'écran de jeu aux neurones de la couche input. Cette technique peut être mise en place pour des IA relatives à des jeux plus compliqués tel que *Super Marios Bros*. Cependant, cette méthode complexifie énormément le perceptron.

Quoiqu'il en soit, avant de rentrer ces données dans le réseau neuronal, il est nécessaire que leurs valeurs respectives soient comprises entre 0 et 1. On effectue donc un « mapping », c'est-à-dire que l'on pondère ces dernières en fonction de la valeur minimum et de la valeur maximum.

¨Par exemple, la distance entre le T-Rex et le cactus étant, de 300 pixels ; la distance maximum étant 600 et minimum 0 ; un mapping pondèrera la valeur 300 en 0,5.

Activation du réseau

À chaque rafraichissement de l'écran de jeu, le réseau neuronal est activé avec les entrées courantes, et sort des valeurs output différentes en fonction des entrées et du poids des connexions.

La sortie

La couche output est constituée de 2 neurones différents. L'un est relié à l'action « sauter » et l'autre à l'action « se baisser ». Lorsque la valeur du neurone de sortie (comprise entre 0 et 1) dépasse un seuil fixé à 0.5, l'action correspondante est déclenchée.

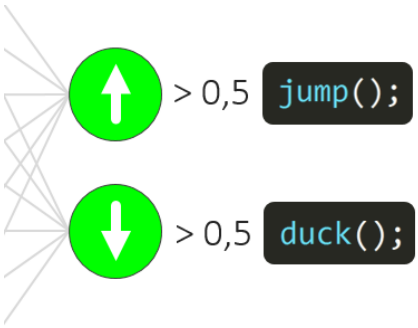


Figure 7 - Lien entre les sorties et les actions

On aurait pu choisir de n'avoir qu'un seul neurone de sortie en reliant l'action « se baisser » si la valeur est inférieure à 0.3, et sauter si la valeur est supérieure à 0.7 (entre 0.3 et 0.7, aucune action n'est déclenchée).

Et où est l'apprentissage dans tout ça ?

Il y a différent moyen d'optimiser un réseau neuronal pour qu'il converge au comportement voulu. Les techniques les plus courantes sont des techniques mathématiques comme la rétro propagation, c'est-à-dire le réajustement des poids du perceptron en fonction de la différence entre la sortie désirée et de la sortie réelle. Cependant, nous n'avons pas l'information sur la sortie réelle attendue (on parle alors d'apprentissage non supervisé). Nous avons alors choisi ici d'optimiser notre perceptron via un algorithme génétique.

Algorithme génétique

Approche générale

Les algorithmes génétiques sont des algorithmes inspirés du processus de sélection naturelle. Ils sont généralement utilisés pour trouver des solutions de bonne qualité à des problèmes d'optimisation, et reposent sur des opérateurs bio-inspirés comme la mutation, le crossing-over ou encore la sélection des meilleurs génomes.

Cependant, il n'est pas si commun que des algorithmes génétiques soient appliqués à des réseaux neuronaux.

Concrètement, un algorithme génétique se déroule selon cette méthodologie :

1. Une génération de N génomes et créée et initialisée aléatoirement,
2. Chaque génome est testé de manière à sélectionner les meilleurs, ceux répondant le mieux au problème,
3. Les meilleurs génomes créent une nouvelle génération de N génomes par crossing-over et mutations
4. La nouvelle génération est testée elle aussi, pour sélectionner les meilleurs. Et ainsi de suite jusqu'à ce que les génomes aient un comportement optimal.

Application au projet

Initialisation

Vous l'aurez certainement compris, les génomes de notre algorithme génétique sont ici les perceptrons étudiés précédemment. Ainsi, dans un premier temps, N perceptrons sont instanciés et initialisés aléatoirement. Comme on peut s'y attendre, leur comportement en jeu est loin d'être cohérent : certains dinosaures sont tout le temps baissés, d'autres sautent en permanence et d'autres encore ne font absolument rien. Cependant, de temps en temps, il arrive que l'un des génomes saute aléatoirement en décalage avec les autres.

Sélection

Le but est alors de sélectionner les meilleurs génomes, c'est-à-dire ceux se débrouillant le mieux face à leur environnement. On va pour cela attribuer un score (appelé *fitness*) à chacun des génomes. L'idée la plus basique consiste à se dire que le score (visible en haut à droite de l'écran de jeu) qu'a établi chaque génome correspond à la *fitness*. Cependant, les obstacles apparaissent de manière aléatoire, alors que les scores augmentent de manière linéaire selon la distance parcourue par notre T-Rex. Ainsi, deux dinosaures aussi mauvais l'un que l'autre pourrait avoir deux *fitnesses* différentes car l'un a rencontré, par chance, un obstacle un peu après l'autre.

Pour répondre à ce problème, on considère alors que la *fitness* doit s'incrémenter en fonction du nombre d'obstacles franchis. De cette manière, si l'un des génomes à une meilleure *fitness* que les autres, il est certain que c'est parce qu'il a bien sauté un cactus.

A l'issue de la course, les deux meilleurs génomes de la génération sont sélectionnés pour être les parents de la prochaine génération. À noter qu'il est possible de modifier le nombre de parents.

Crossing-over

Arrive ensuite la phase de crossing-over. À la manière de deux chromosomes se scindant aléatoirement en leur centre pour mélanger leurs ADN respectifs, les perceptrons des deux génomes parents vont être scindés en deux parties afin de créer un nouveau perceptron possédant un mélange entre les neurones et les poids de ses deux parents.

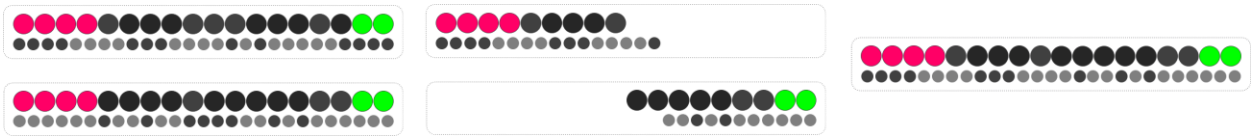


Figure 8 - Crossing-Over : Avant - Pendant - Après

Mutations

Une fois la phase de crossing-over franchie, on applique des mutations aléatoires sur les génomes. Ces mutations sont un passage obligé si l'on veut que nos enfants aient une diversité de comportement plus vaste que celle proposée dans la génération initiale.

Dans le cas de nos perceptrons, la mutation est concrètement implémentée sous la forme d'un changement mineur d'une des valeurs du perceptron, relative à un neurone ou à un poids.

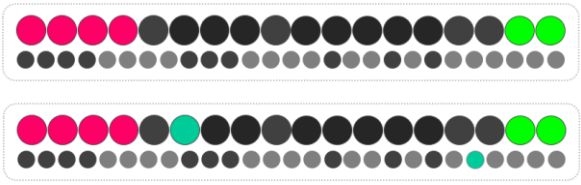


Figure 9 - Mutation : Avant - Après

Les mutations ont lieu selon une probabilité fixée à l'avance, dans notre cas, la valeur de la probabilité est fixée à 0.2%. Contrairement à ce que je pouvais penser, augmenter drastiquement ce nombre ne permet pas de converger plus rapidement vers un comportement cohérent.

La variation de la mutation est elle aussi fixée à l'avance, ici, la valeur mutée est multipliée par un facteur choisi aléatoirement entre 0.5 et 1.

Amélioration de la vitesse de convergence

Une fois l'algorithme en place, Les génomes convergent tout doucement vers un résultat cohérent. Jusqu'à présent, la vitesse de convergence n'avait pas vraiment d'importance, l'objectif étant de réussir à ce que nos dinosaures apprennent d'eux-mêmes à jouer au jeu. À ce stade là, la partie la plus compliquée du projet est franchie avec succès. Ensuite vient la phase d'amélioration de l'algorithme existant. Il s'agit alors de réduire le temps de convergence entre la génération 0 et une génération optimale, jouant plus ou moins comme un humain.

Pour cela, plusieurs techniques ont été mises en place.

Contrer la régression

La première chose remarquable est que, parfois, l'ensemble des enfants d'une génération a un comportement bien plus mauvais que celui des parents.

Pour éviter ce problème, j'ai créé des clones d'élites, c'est-à-dire que j'ai replacé des clones des deux meilleurs parents dans la génération suivante. Ainsi, je m'assure que le patrimoine génétique de la génération courante a, dans le pire des cas, le comportement des parents.

Changement des paramètres

Comme on peut s'en douter, modifier les paramètres de l'algorithme génétique influe fortement sur la vitesse de convergence. Parmi les paramètres modifiables, on retrouve, entre autres :

- Le nombre N d'individus par génération
- Le nombre de parents
- Le taux de mutation
- Le taux de variation des mutations

Ici, le résultat le plus probant fut l'augmentation du nombre de génomes par génération. De 12, je suis passé à 40, réduisant énormément le temps de convergence.

Les autres paramètres n'ont pas une influence aussi importante.

Calcul de la fitness

Comme expliqué précédemment, lorsqu'un T-Rex franchi un obstacle, sa *fitness* est incrémentée. Cependant, avec cette méthode, certains individus ne faisant que sauter en permanence arrivait à avoir une meilleure *fitness* que d'autres tout aussi mauvais. En soit, il s'agissait bien des plus intelligents de leur génération puisque sauter en permanence augmente effectivement les chances de sauter un obstacle. Mais cela entraine une évolution de l'ensemble des génomes vers un optimum local, ralentissant par là-même la vitesse de convergence vers le comportement optimal final.

Pour contourner ce problème, la fonction de *fitness* a été réécrite de manière à ce que la *fitness* correspondent non plus aux nombres d'obstacles sautés, mais au rapport entre le nombre d'obstacles sautés et le nombre de sauts effectués en tout. Ainsi, un individu ne faisant que sauter est désavantagé par rapport à un individu ayant sauté seulement aux bons moments.

On passe de : $fitness = jumped$ à une formule améliorée : $fitness = jumped * \frac{jumped}{jumps}$

Cependant, il existe encore de nombreux axes d'améliorations possibles, et nous en verrons quelques-uns dans la partie Pistes d'améliorations.

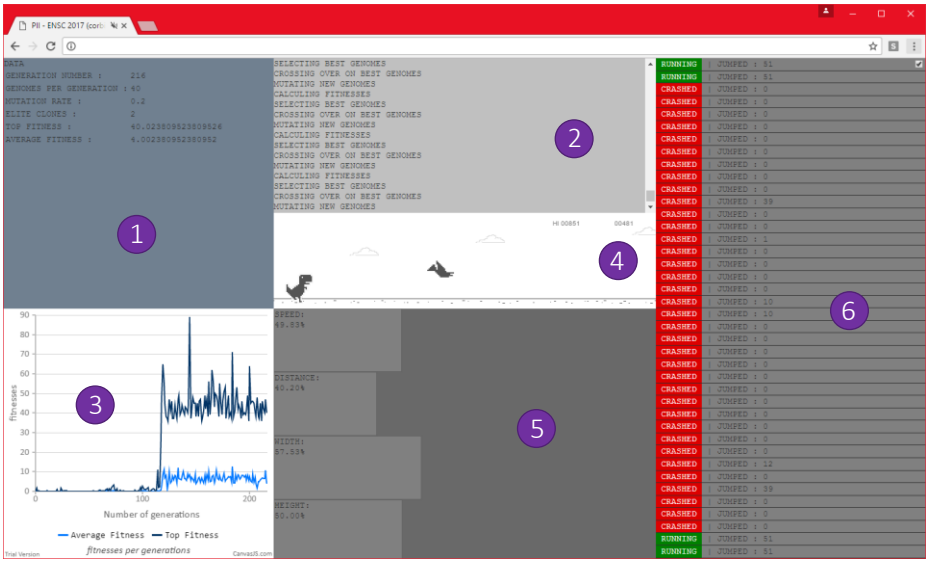
Interface finale

Une fois l'objectif atteint (lorsque le T-Rex apprend assez rapidement), une interface a pu être élaborée autour du bloc de jeu. Il s'agit d'une interface donnant de multiples informations relatives à l'exécution du programme.

Coté responsif

L'interface s'adapte à différentes tailles d'écran (au chargement de la page et au redimensionnement). Ce fut un petit challenge en soi étant donné que le bloc de jeu a une taille fixée (600 x 150). Ainsi, les tailles des autres blocs doivent s'adapter en fonction du bloc central. L'aspect responsive a été réalisé principalement grâce à la librairie JQuery, une surcouche de Javascript permettant d'en utiliser les fonctions plus simplement.

L'interface se présente comme affichée ci-dessous :



1. Données globales
2. Historique des opérations
3. Graphique des *fitness*
4. Jeu
5. Inputs
6. Détail des génomes

Figure 10 -Interface numérotée

Détail des blocs de l'interface

Données globales

Ici sont présentées les données globales sur les variables utilisées dans l'algorithme génétique.

GENERATION NUMBER	Numéro de la génération courante
GENOME PER GENERATION	Nombre d'individu en course à chaque génération
MUTATION RATE	Probabilité de muter les génomes enfant à chaque nouvelle génération
ELITE CLONES	Nombre de clones d'élites (meilleurs génomes conservés dans la génération suivante pour éviter toute régression)
TOP FITNESS	*Fitness* maximum de la génération passée
AVERAGE FITNESS	Moyenne des *fitness* de la génération passée

Historiques des opérations

Ce bloc affiche certains messages lors de l'exécution du code. Ces messages sont notamment relatifs aux étapes successives de l'algorithme génétique. L'historique des opérations permet de garder une trace des événements tout en scrollant automatiquement sur les derniers messages affichés.

Graphique des fitness

Il s'agit probablement ici du bloc le plus pertinent. Ce graphique présente l'évolution de la meilleure *fitness* et de la moyenne des *fitness* au fil des générations.

La librairie utilisée est Canvas.js, qui permet de redessiner son graphique en temps réel.

On remarque graphique des meilleures *fitness* (en bleu foncé) deux choses :

1. Il n'est pas lisse.

Cela s'explique par le caractère aléatoire de l'environnement. Si l'environnement était identique à chaque génération, le graphique serait complètement lisse.

2. On y voit un palier

Ce palier correspond à l'étape que franchi le perceptron en apprenant à sauter un cactus. La courbe finale est constituée de plusieurs paliers, avec notamment celui de l'apprentissage de la réaction à avoir face à un ptérodactyle (ces derniers apparaissent de manière aléatoire si le score est au minimum à 350).

Jeu

Il s'agit ici de l'interface centrale. À noter qu'il est possible de cacher/afficher l'interface autour avec la touche `I` (comme interface), ou en cliquant sur la discrète checkbox située en haut à droite de la page.

Inputs

Ici, une représentation graphique des valeurs qui entrent dans le réseau est visible. Ces entrées sont les inputs correspondant aux capteurs. Ils sont égaux pour tous les T-Rexs étant donné que chaque génération évolue dans le même environnement.

Bien que les valeurs soient réellement comprises entre 0 et 1, on affiche leur valeur multipliée par 100 pour plus de lisibilité.

On observe que leur rafraichissement est directement corrélé au rafraichissement du jeu.

Détail des génomes

Sur le côté droit, on retrouve une liste de tous les génomes en course avec leur statuts (RUNNING ou CRASHED). On peut également voir le nombre d'obstacles franchis.

Ici, on rafraichit localement chaque statut lorsque cela est nécessaire afin de gagner en performance.

Pistes d'amélioration

Réduction de la vitesse de convergence

Bien qu'un travail sur la vitesse de convergence des réseaux neuronaux vers un comportement optimal ait été réalisé tout au long du projet, il est possible d'user d'autres solutions toujours plus astucieuses.

Parmi elles, celle qui me paraît la plus prometteuse porte sur la phase de crossing-over. Actuellement, les deux meilleurs parents partagent leur génome respectif de manière aléatoire. Avec ce système, il arrive parfois qu'un parent A ait une excellente fitness alors qu'un parent B a une fitness quasiment nulle. Le crossing-over donnera généralement des enfants avec un comportement entre les deux : ni très bon, ni très mauvais, ce qui est dommage puisque s'ils avaient plus hérité du parent A, ils seraient meilleurs.

Une solution à ce problème serait d'effectuer un crossing-over proportionnel aux *fitness* : plus le génome a été bon, plus ses gènes auraient de chances d'être transmis aux enfants.

Amélioration de l'interface

L'interface peut clairement être améliorée sur le plan ergonomique comme sur le plan esthétique.

Gestion de projet

CDC

La première version du cahier des charges avait le double but d'une part, de présenter son projet à l'équipe enseignante afin qu'il soit validé ou non, et d'autre part, d'avoir soi-même une bonne idée globale des objectifs de notre projet ainsi que des contraintes qui lui sont liées.

Comme tout cahier des charges, il est censé être évolutif et doit préciser au fur et à mesure du projet les fonctionnalités attendues. Cependant, dans le cadre de ce projet précis, je ne l'ai honnêtement jamais retouché. Cela s'explique car, toujours dans ce cadre précis, un cahier des charges n'était pas vraiment pertinent, et cela pour deux raisons principales :

1. L'objectif de mon projet est plutôt binaire : le but était de réussir à avoir une Intelligence Artificielle qui apprend. Il n'existe donc pas une liste exhaustive de fonctionnalités détaillées à implémenter.
2. Par ailleurs, le projet est de petite envergure et exécuté en monôme. Il est donc aisé de garder en tête l'ensemble des contraintes et du travail à faire au fur et à mesure du temps ; d'autant plus que l'objectif est unitaire.

Gantt

Le diagramme de Gantt est un élément tout aussi évolutif et permet de savoir où l'on en est dans la réalisation du projet. Cependant, une fois la version initiale établit, je ne l'ai pas retouché non plus. Il m'est toutefois arrivé de le consulter à plusieurs reprises afin d'avoir l'information sur les limites de dates. Ici aussi, l'exécution du projet en monôme permet de garder une idée claire de son avancement, d'autant plus lorsque l'on retouche au projet quasiment tous les jours.

GitHub

Bien que GitHub serve habituellement à partager du code entre développeurs de manière simple via un serveur distant, il s'est avéré bien utile quant à l'organisation de ce projet.

D'une part, chaque commit est archivé et permet alors d'avoir une vision du travail effectué, et d'autre part, les statistiques établies par le site sur le projet permettent d'avoir une preuve concrète du travail fourni via un ensemble de graphiques.

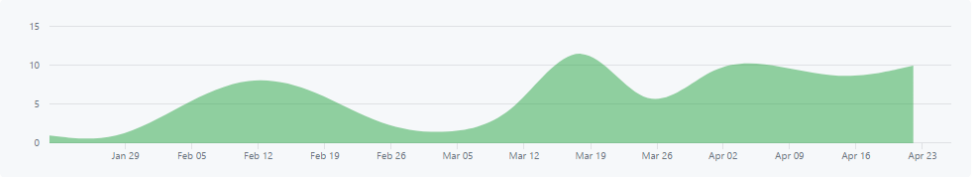


Figure 11 - Graphique issu de github.com

Ce premier graphique présente l'activité sur le projet au cours du temps. La forme de cette courbe montre un travail plutôt régulier, via trois longues vagues successives correspondant respectivement au lancement du projet, au travail fourni pour arriver à avoir une IA qui évolue, et enfin au soin apporté à l'interface entourant le projet. Parfois, les projets se finissent plutôt de manière exponentielle, ce qui n'est pas le cas ici.

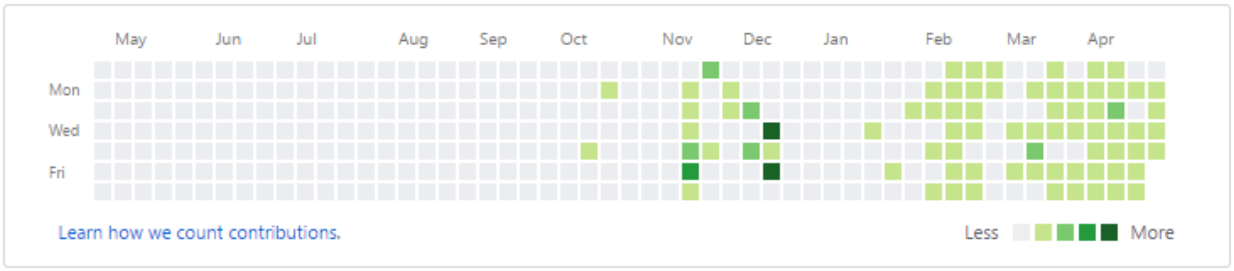


Figure 12- Graphique issu de github.com

Ce second graphique présente la fréquence des *commits* pour chaque jour de l'année. Chaque case correspond à un jour, et sa couleur correspond à l'intensité du travail (vert clair : faible jusqu'à vert foncé : importante). Ici, comme le projet a commencé début février, on comprend qu'il s'est déroulé sous la forme d'un marathon plutôt que sous la forme d'un sprint.

Difficultés rencontrées

Une barre plutôt haute

Malgré une obligation de résultat (il faut bien un résultat concret à la fin), j'ai pris le parti ambitieux de tenter un projet que je n'étais pas sûr de réussir. Je préfère fixer la barre haute et ne pas l'atteindre plutôt que de réaliser un projet sur un domaine que j'ai déjà étudié.

En effet, pour que l'objectif soit rempli, il fallait que j'aie une IA qui apprenne. Le résultat est binaire : soit c'est le cas, soit ça ne l'est pas. J'appréhendais particulièrement la phase de « vérité », celle où j'ai testé le code censé être fonctionnel. Si l'IA n'avait pas appris à ce moment là, le problème aurait été très compliqué à résoudre. Il n'aurait pas été question d'une ligne de code à reprendre, mais bien d'une vérification de l'ensemble de la théorie et son application pour débusquer l'erreur de raisonnement.

Quoiqu'il en soit, mon premier réflexe a été de recoder le jeu en Javascript (langage peu connu jusqu'alors). C'était là une manière de me protéger en cas d'échec de l'objectif.

Une librairie empoisonnée

La librairie P5 a été l'un des problèmes techniques majeures de mon projet. En effet, cette dernière semblait parfaite pour recoder le jeu, l'environnement du code permettant déjà un rafraîchissement de l'écran se déclenchant automatiquement. Aussi, les fonctions annexes d'affichage d'images, du texte, et ainsi de suite étaient bien pratiques. Cependant, les problèmes sont arrivés lorsque j'ai voulu intégrer mon code (du jeu) dans mon algorithme génétique. Les classes ne fonctionnait pas entre elles (l'architecture de la librairie ne le permettant pas !).

Ainsi, j'ai recodé les fonctions de P5 qui posait problème comme la fonction de rafraichissement ou celle de chargement des images. J'ai cependant gardé la librairie pour pouvoir utiliser les fonctions mathématiques utiles que je n'avais pas le temps de recoder.

Langages rencontrés

Ce projet fût la découverte du langage Javascript, langage dominant du Web coté client. Le langage n'est pas très rigoureux mais il est probable que malgré cela, il devienne l'un des langages les plus populaires dans le futur proche, surtout avec les objets connectés.

JQuery fût aussi une découverte. Bien qu'il s'agisse d'une librairie, sa logique d'utilisation est unique et le langage reste utilisé très fréquemment dans le domaine du web également.

Conclusion

Je suis globalement plutôt fier de ce projet. Les notions de réseaux neuronaux et d'algorithmes génétiques sont très motivants, et la réussite du projet réussite me confirme que j'ai fait le bon choix de sujet. J'ai pu apprendre le Javascript de manière assez approfondie, et utiliser de nombreuses librairies que je reverrais surement dans le futur.

Le projet est également accessible depuis un navigateur, je peux donc l'intégrer facilement à un portfolio des réalisations que j'ai imaginées lors de mes études, ou dans un éventuel CV en ligne.

J'ai réussi ce projet avec un minimum d'aide, de la jugeotte et une bonne dose de travail. Enfin, si je devais choisir un sujet pour un second Projet Informatique Individuel, ce serait probablement un approfondissement de la théorie des réseaux neuronaux, tout simplement passionnante.