Rapport de Projet Info

Algorithme génétique appliqué aux réseaux neuronaux

Tableau des refs

# Introduction

## Problématique

Le projet consiste ici 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) du type T-rex de Google Chrome. 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 des cours 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 relatif aux problématiques adjacentes, j’avais l’ensemble des informations nécessaires pour me lancer dans le bain de ces concepts a priori plutôt abstraits.

## Pré-existant

### Le jeu

Le premier élément existant est le jeu en lui-même. Accessible depuis l’interface de Google Chrome et codé 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 à moi. Quoiqu’il en soit, les sprites (images utilisées dans le jeu), les sons, et d’autres éléments ont pu être repris par la suite.

### Le travail d’Ivan Seidel

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, et, surtout, Ivan Seidel, un brésilien ayant fait une vidéo de vulgarisation sur ce sujet même.

# 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 faut ensuite appuyer sur la barre espace.

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

Le but du jeu 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](https://www.youtube.com/watch?v=g5U8WyBPjt4).

## Architecture logicielle

### Diagramme de classe

Le code est organisé en une page HTML, une page CSS et un script de lancement en Javascript auxquels s’ajoutent neuf autres classes javascripts comme présentées dans le diagramme ci-dessous :

DIAGRAMME DE CLASSE

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 tel que présentée ci-dessous.

DIAGRAMME ARBORESCENCE FICHIER

### 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 fonction très pratiques pour afficher les sprites ou encore de nombreuses fonctions mathématiques bien pratiques. Cependant, cette librairie a posé de nombreux 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 :

EXEMPLE ICI

Cependant, cet 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 pour exploiter la théorie de manière pertinente. Il s’agit d’une théorie mathématique bio-inspirée, qui singe le fonctionnement des neurones dans un cerveau biologique tel que nous le connaissons.

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

FIGURE ICI

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. 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 développeur. Le nombres 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 imposée plus ou moins d’elle-même.

En effet, il aurait été possible de recoder soi-même les concepts de perceptrons, neurones et connexions, mais lors du choix, l’objectif principal était d’arriver à un résultat fonctionnel (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 devoir réinventer la roue.

### Concrètement, ça donne quoi ?

Le perceptron associé à chaque T-Rex se base sur le modèle suivant.

GRAPHIQUE

#### 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 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. Un chercheur dans le domaine expérimenté trouvera plus facilement, avec son savoir et son intuition, quel est le meilleur paramétrage pour la création de son perceptron. Cependant, il faut 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

Dans notre projet, chaque Dinosaure possède un Brain associé à un Genome lui-même relié à un Perceptron.

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).

GRAPHIQUES INPUTS

Dans le cas où l’on ne connait pas l’ensemble des paramètres nécessaires au T-Rex pour apprendre à jouer, il est possible de relier chaque pixel de l’écran de jeu à la couche input. Cette technique peut être mise en place pour 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 leur valeur soit comprise entre 0 et 1. On effectue donc un « mapping » de ces valeurs, c’est-à-dire qu’on les pondère en fonction de la valeur minimum et de la valeur maximum.

Ainsi, la distance entre le T-Rex et le cactus étant, par exemple 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 ».

GRAPHIQUE

On aurait pu choisir de n’avoir qu’un seul neurone de sortie et relier les actions de la manière suivante, qui aurait fonctionné également.

GRAPHIQUE

#### 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. La technique les techniques les plus courantes sont des techniques mathématiques comme la retropropagation, c’est-à-dire le réajustement des poids du perceptron en fonction de la sortie désirée et de la sortie réelle. Cependant, nous avons choisi d’optimiser notre perceptron via un algorithme génétique.

# Algorithme génétique

## Approche générale

* Bio inspirés
* Principe de base

## Application au projet

* Mais avec une génération de N individus, comment on fait avec le jeu de chrome qui ne permet de jouer qu’un seul T-REX à la fois ?
  + On recode le jeu
    - Bien que le code du jeu existant soit extractible, c’est 2000 lignes de code peu compréhensibles
    - En recodant, on a un bien meilleur contrôle de notre jeu (on peut mettre 10 Trex dans la même course, c’est plus facile de relier les outputs du réseau neuronal aux actions de nos trex, etc)
* Application au jeu + schémaS
* Choix des paramètres
* Amélioration au fur et à mesure en changeant les paramètres pour améliorer la vitesse de convergence vers un comportement satisfaisant
  + Long à tester à chaque changment

# Interface finale

* Une fois l’objectif atteint, à savoir : avoir un trex qui apprend, on a créé une interface autour, donnant de multiples informations sur l’état courant de l’exécution de l’algorithme
* Captures d’écran légendée

## Coté responsif

* L’interface s’adapte à différentes tailles d’écran au chargement de la page et au resizing
* Léger challenge car le canva de jeu a une taille fixée (600 \* 150) donc les autres blocs doivent s’adapter en fonction de lui
* Schéma (captures d’écran 1080 et 720)

## Détail des blocs de l’interface (mal dit)

### Données globales

* Données globales sur les variables utilisées dans l’algorithme génétique
* Tableau détaillé VAR + EXPLICATION

### Historiques des opérations

* Garde une trace de ce qui s’est passé au fur et à mesure de l’exécution de l’algorithme génétique, garde l’historique et scroll automatiquement sur les derniers événements

### Graphique des fitness

* Probablement le bloc le plus pertinent
* Utilisation de la librairies Canvas.js
* Évolution de la fitness maximum au fil des générations et de la fitness moyenne de la génération
* Schémas au fil de quelques générations
* Explication des paliers, avec droites paliers superposées à la capture d’écran
* Si le terrain était fixe, le graphique serait comme le deux car le clone d’élite de la génération précédente se comporterait exactement de la même manière face à l’environnement identique

### Jeu

* Le jeu qu’on a recodé avec canvas p5

### Inputs

* Inputs communs à tous les génomes en course : se rafraichissent à une fréquence de 60 fps, comme les inputs eux même
* Inputs entre 0 et 1 mais multipliés par 100 pour plus de clarté
* Représentation graphique de leur changement

### Détail des génomes

* Détail de N génomes en course : leur statuts (vivant ou mort) + le nombre de sauts qu’ils ont effectués
* Se rafraichit de manière localiser pour gagner en performance

# Pistes d’amélioration

## Réduction de la vitesse de convergence

* Bien qu’un travail sur la vitesse de convergence des RN vers un comportement optimal ai été réalisé tout au long du projet, il est possible d’user d’autres solutions astucieuses.
* Parmi elle, ma préférée est la transmission de gènes d’un parent proportionnellement à sa fitness
  + Si deux parents : un nul et l’autre parfait , alors enfant moyen, ce qui est dommage
  + Schéma
* Simplification du Réseau neuronal, mais peut prometteur

## Amélioration de l’interface

* Ergonomie
* Esthétique

# Gestion de projet

## CDC

* But : Valider ou pas le projet par l’équipe enseignante
* Avoir une idée de ce qu’on veut à la fin
* Censé être évolutif mais je l’ai pas retouché
* Justification : un CDC dans le cadre de mon projet était peu pertinent
  + Objectif binaire, pas une liste bien précise de contraintes et de fonctionnalités à implémenter
  + Projet de petite taille et exécuté en monôme : on arrive à garder tout en tête

## Gantt

* Avoir une idée de comment on va repartir ses tâches dans le temps
* Il m’ait arrivé de le regarder pour savoir si j’étais vraiment en retard où pas, mais cela se compte sur les doigts d’une main
* Projet de petite taille et exécuté en monôme, on sait exactement ce qui a été fait et ce qu’il reste à faire à tout moment…

## GitHub

* Cela ressemble plus à un élément technique, mais non cela m’a bien servi pour la gesp
  + Vision globale du travail effectué
  + Preuve de travail via graphiques
* Schéma graphs : régulier et non pas exponentiel
* Schéma tableau des commit : petites charges de travail réparties uniformément
* Un marathon et pas un sprint

# Difficultés rencontrées

## Une barre plutôt haute

* Je préfère fixer la barre haute et rater que me reposer sur mes lauriers, puis c’est le but du PII de toutes façons
* Pour que le projet soit pertinent il faut que ça marche, mais l’objectif est compliqué à atteindre, et plus ou moins binaire (ça marche ou ça ne marche pas), du moins jusqu’à ce que le trex apprenne
* Si ça ne marche pas, c’est super compliqué à comprendre d’où vient le problème car ça veut dire que j’ai mal appliqué la théorie…
* Pour se protéger, on a recodé le jeu en javascript (ce qui aurait déjà pu être un projet en soit étant donné mon niveau nul en javascript avant le projet)
* Recoder le jeu : permet de gérer le nombre d’individus par génération, et meilleur contrôle sur les actions (rejouer, sauter, se baisser, lancer une partie, etc)

## Une librairie empoisonnée

* Recodage du jeu de chrome assez rapides mais d’énorme problèmes ensuite lorsqu’il a fallu intégrer à cela l’algorithme génétique et le second canvas pour le graphique
* Solution : recoder moi-même les fonctions problématiques de la boucle de jeu
* On a laissé la librairie car on se resservait de fonctions mathématiques bien pratiques, comme random, map, etc qu’on avait pas le temps de recoder

# Langages rencontrés

* Javascript
  + P5 (Processing)
* JQuery