CFPT-INFORMATIQUE

Travaux de Diplômes 2017

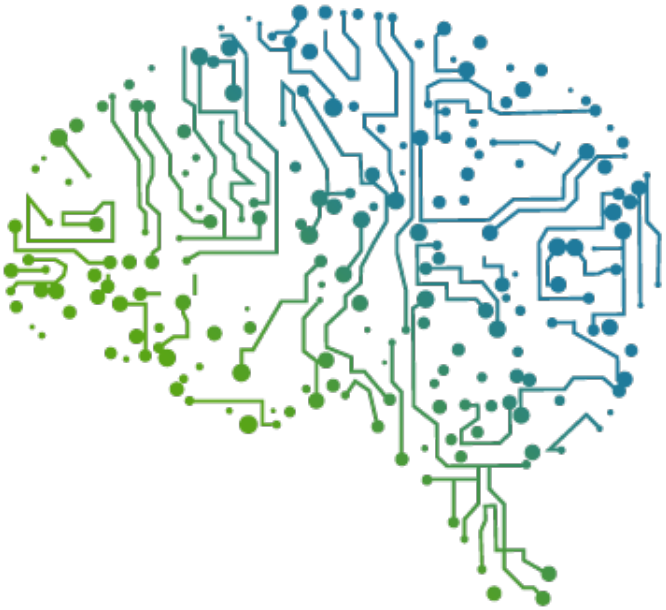
T-Rex Ceptional

Neuroevolution

Loris De Biasi

Supervisé par :

M. Wanner



T.IS-E2A

12 juin 2017

1 Résumé

Le but de ce projet est de créer une *IA* qui a pour objectif d'apprendre à jouer à plusieurs jeux vidéo. Il implémente des principes du *Machine learning* tel que les réseaux de neurones, ainsi que les algorithmes génétiques.

Un réseau de neurones de type *supervised* permettra de créer la structure du cerveau de l'*IA*, alors que les algorithmes génétiques permettront de l'entraîner. Chaque cerveau sera envoyé dans une simulation d'un jeu et obtiendra un score lorsqu'il meurt. Ce score sera utilisé pour déterminer lesquels sélectionner et "reproduire" entre eux pour ainsi donner des cerveaux plus "intelligent".

le réseau de neurones prend des valeurs normalisées du jeu en entrée, puis, les valeurs d'entrées ainsi que des "poids" (nombres) passent dans des fonctions mathématiques pour déterminer l'action à effectuer. Une fois que chaque réseau a été utilisé dans une simulation, on prend ceux qui ont le plus gros score pour interchanger leur "poids" et les modifier légèrement. Ces opérations sont répétées infiniment.

Ce projet permet d'ajouter des jeux en renseignant quelques valeurs et obtient de très bon score sur les jeux déjà implémentés.

2 Abstract

The purpose of this project is to create an *Artificial intelligence* that aims to learn to play several video games. It implements some principles of *Machine learning* such as neural networks, as well as genetic algorithms.

A neural network of the *supervised* type will allow the creation of the structure of the brain used by the *Artificial intelligence*, while the genetic algorithm will train it. Each brain will run in a simulation of a game and will get a score when the game ends. This score will be used to select the best brains. Those brains will then reproduce and "crossover", thus merging and mixing values.

The neural network takes normalized values and inputs from the game as well as "weights" that pass into mathematical functions. Those functions determine what action to make. Once each neural network has been used in a simulation, we take the ones with the biggest score to interchange their "weight" and modify them slightly. Thoses operations are repeated infinitely.

This project allows to add new games by refering some values and gets a very good score on the games already implemented.

Table des matières

3 Cahier des charges

3.1 Titre du projet

Intelligence artificielle apprenant à jouer à un jeu vidéo et ayant pour but d'aller le plus loin possible.

3.2 Objectifs du projet

Création d'une *IA* ayant pour objectif d'apprendre à jouer au *T-Rex* runner de chrome.

- Aucune bibliothèque ne sera utilisée.
- Affichage du jeu avec l'*IA* en direct.
- L'*IA* ne connaît pas les règles du jeu.
- L'*IA* connaît les touches permettant d'effectuer une action, mais ne connaît pas l'action effectuée.
- Affichage des touches pressées par l'*IA*.

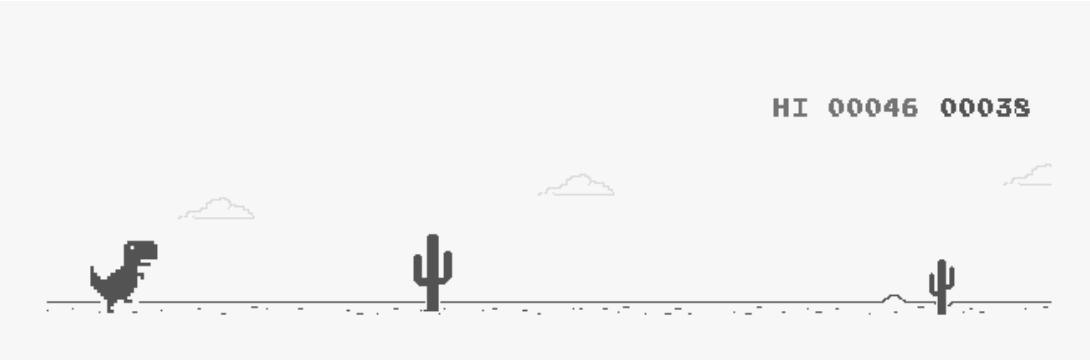


FIGURE 1 – Jeu du T-rex

3.3 Description détaillée

Le but est de créer une *IA* qui aura la capacité d'apprendre à jouer au jeu *T-rex runner* présent sur le navigateur *Google chrome* lorsqu'il n'y a aucune connexion à internet.

Le principe de ce jeu est simple, vous contrôlez un personnage (*T-Rex*) qui avance automatiquement. Devant vous apparaissent soit des cactus (obstacle au sol), soit des ptérodactyles (ennemi dans les airs). Les seules actions possibles sont soit de se baisser, soit de sauter. Le but est donc d'arriver le plus loin sans se faire toucher une seule fois.

L'*IA* utilisera deux principes, le *Machine learning* et les algorithmes génétiques.

Le *Machine learning* sera là pour créer un "cerveau" (réseau de neurones) à cette *IA*, lui permettant ainsi de définir des règles. Dans ce cas, il ressemblera à quelque chose similaire à cela :

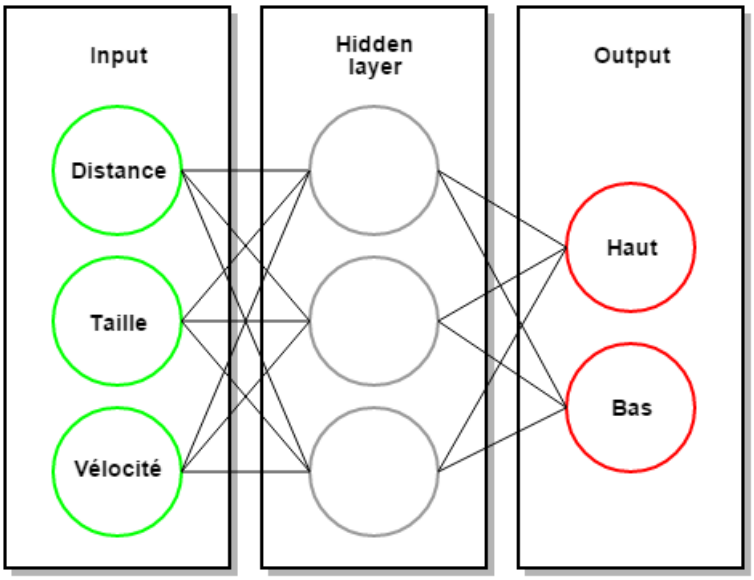


FIGURE 2 – Exemple du réseau de neurones du *T-Rex*

Dans le rectangle de gauche se trouve la zone "input", permettant de définir quelles caractéristiques du jeu influeront sur les actions du *T-Rex*. Dans ce cas, trois choses sont requises :

- **La distance** : permettant de connaître la distance séparant le *T-Rex* d'un obstacle
- **La taille de l'obstacle** : permettant de connaître la taille en largeur de l'obstacle le plus proche
- **La vélocité** : permettant de connaître la vitesse actuelle du *T-Rex*.

Lorsque le *T-Rex* court, ces trois caractéristiques sont requises, car si le *T-Rex* ne connaît pas la distance le séparant de l'obstacle, il ne saura jamais quand sauter. De même si le *T-Rex* ne connaît ni sa vélocité ni la taille de l'obstacle, il ne comprendra jamais qu'il faut un timing précis pour un obstacle plus long ou bien avec une vitesse plus élevée.

Dans un premier temps, ces informations seront récupérées directement en ayant accès au code, mais il est envisageable de le faire en examinant le jeu, sans avoir directement accès à celui-ci.

Dans le rectangle du milieu se trouve la zone "Hidden *layer*". Cette zone contient une seule chose, des neurones. Ces neurones sont en fait simplement des fonctions qui prennent en entrée un nombre, puis le passe dans la fonction et le transmet, soit au neurone suivant, soit à la troisième zone ("output"). Ces fonctions sont appelées "fonction d'activation" et ils en existent énormément de types différents, permettant toute quelque chose de différent. Les plus connus étant : Sigmoig, ReLu, Tanh et Linear. Le nombre de neurones est-ce que l'on appelle un *hyperparameter*, cela signifie qu'ils ne peuvent pas être appris par le réseau de neurones et qu'il faut les renseigner à la main. Les neurones présents dans l'image du haut ne sont qu'un exemple, le réseau de neurones final ne ressemblera probablement pas à cela et contiendra certainement plus de neurones ainsi que plusieurs "Hidden *layer*". Plus un réseau est complexe, plus il permet d'avoir un résultat précis, mais plus il prend de temps, il faut donc choisir le bon ratio temps/résultat.

Avant de parler du troisième rectangle, il faut savoir que tous les traits entre les cercles représentent des "poids", il s'agit d'un nombre que l'on va utiliser avec le nombre du neurone précèdent pour la passer dans la fonction d'activation. Ces "poids" sont les seules choses qui varient dans un réseau de neurones "standard".

Finalement, la troisième partie, appelé "Output", est simplement le résultat de tous les calculs précèdent. Selon le résultat obtenu sur le neurone de sortie, la touche sera pressée ou non. Par exemple, si le résultat de sortie est "0.6" et que nous avons défini que la touche s'active seulement si un nombre supérieur à "0.5" est trouvé, alors la touche sera activée. Il faut savoir que les résultats de sortie de chaque neurone sont "normalisés" en général entre 0 et 1, grâce aux valeurs minimales et maximales du réseau de neurones.

Parlons maintenant de l'algorithme génétique. Il permet de créer plein de réseau de neurones, appelé "Genome". Comme dis précédemment, la seule variable d'un réseau de neurones sont ses poids, nous allons donc créer plein de réseau de neurones avec des poids différents. Ces réseaux de neurones sont ensuite testés sur le jeu, puis, en faisant de la sélection naturelle, nous allons jeter ceux ayant le moins bon score. Ensuite, nous allons faire ce que l'on appelle de l'enjambement ("cross over" en anglais). Cela consiste à prendre des parties de chacun des réseaux de neurones restant pour les combiner en un seul. Par la suite, nous allons choisir des valeurs aléatoires auxquelles seront ajoutées des valeurs aléatoires (étape appelée "mutation"). Il ne reste plus qu'à répéter ce processus jusqu'à avoir le nombre de génome souhaité, créant ainsi une nouvelle génération. Tout ce processus (sélection, enjambement et mutation) est répété jusqu'à avoir le résultat escompté.

3.4 Inventaire des étapes du projet

Le total des heures correspond à 312 heures (39 x 8h00).

Début : Mercredi 5 avril 2017
Reddition intermédiaire (documentation + poster) : Vendredi 5 mai 2017
Reddition intermédiaire (résumé + abstract) : Vendredi 19 mai 2017
Reddition finale : Lundi 12 juin 2017

3.5 Inventaire du matériel

— PC + 2 écrans
— clavier USB
— souris USB

3.6 Inventaire des logiciels

— Système d'exploitation : *Windows 10* ou *Linux mint*
— Outils de développement : *Sublime text 3*
— Langage de programmation : *Javascript/HTML/CSS*

3.7 Délivrables

— 1 carnet de bord

— 1 exemplaire papier de la documentation technique

— 1 exemplaire papier du mode d'emploi

— 1 CD contenant les sources (projet logiciel)

— d'autres exemplaires papier de la documentation technique et du mode d'emploi en fonction des demandes des experts

— mis à part le carnet de bord, tous les documents seront également restitués sur le serveur *Moodle*

4 Planification prévisionnelle

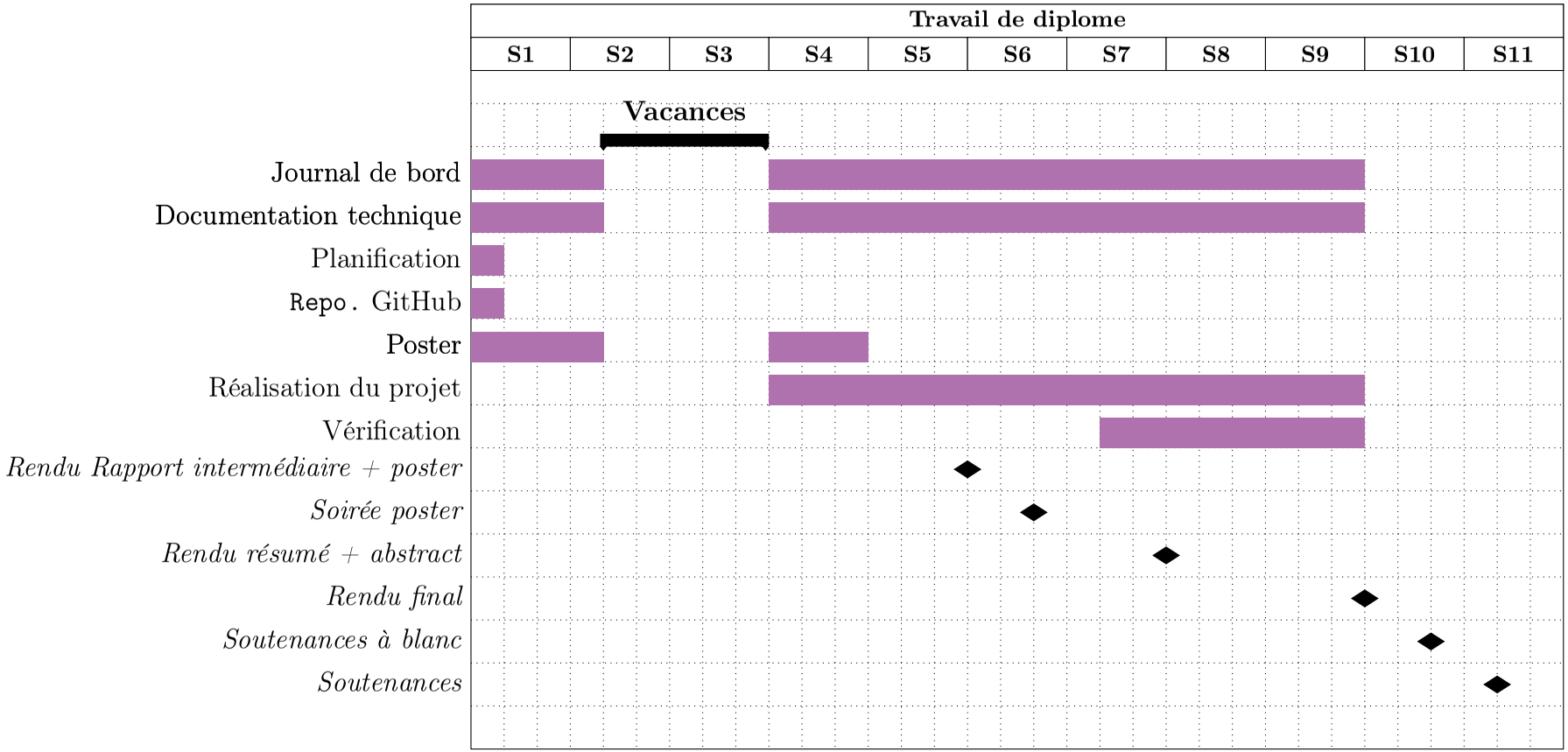


FIGURE 3 – Diagramme de Gantt (prévisionnelle)

5 Planification réel

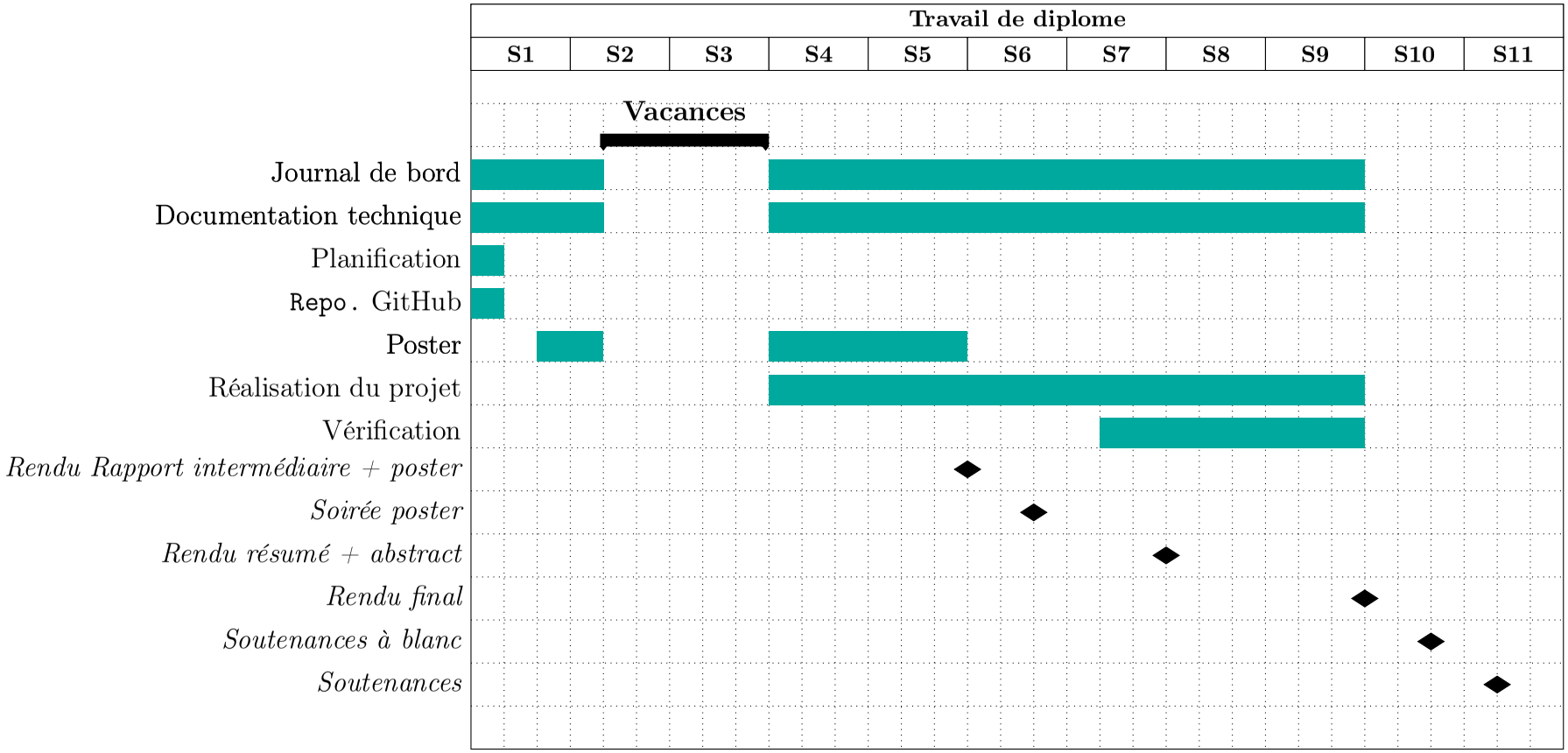


FIGURE 4 – Diagramme de Gantt (réel)

6 Introduction

Le *Machine learning*[1] est devenu de plus en plus important et de plus en plus utilisé ces dernières années. Un exemple concret serait les voitures automatiques dont les constructeurs n'arrêtent pas de vanter les mérites, Google adWords[2] qui génère plus de 60 milliards de dollars par année ou bien encore les moteurs de recommandations telles que celui d'Amazon ou de Spotify qui permettent d'avoir des recommandations selon ce que vous écoutez/regardez.

Mon projet aura pour but de me plonger dans ce concept pour ainsi apprendre son fonctionnement et comprendre plus en profondeur ses mécanismes. Pour cela, j'ai choisi de développer une *IA*[3] pour le jeu du *T-Rex* présent sur *Google Chrome* lorsque qu'aucune connexion à internet n'est disponible. Pour être plus précis, je ne vais pas créer une *IA*, mais plutôt créer un cerveau "vierge" qui, sans avoir connaissance des règles du jeu, devra apprendre de lui-même.

Étant donné que je souhaite apprendre un maximum de choses, je ne me servirai d'aucune bibliothèque lors de ce projet.

7 Étude d'opportunité

L'utilisation du *Machine learning* dans les jeux vidéo est quelque chose de très peu pratiqué et cela est dû à plusieurs raisons. Premièrement, une *IA* utilisant le *Machine learning* ne peut pas être contrôlée, elle effectuera les actions qu'elle aura apprises, la rendant ainsi imprévisible. De plus, que ce soit pour le développement ou lors de l'exécution, cela n'est pas rentable en termes de temps. Elles sont plus complexes à développer et plus lente à l'exécution, mais permettent d'obtenir des résultats qui seraient impossibles à coder "a la main".

Néanmoins, il existe tout de même des jeux vidéo utilisant le *Machine learning*. Il s'agit cependant, très souvent, de personne qui développe par-dessus des jeux existants. En voici quelques exemples.

1. "apprentissage automatique" en français
2. régie publicitaire de Google
3. "dispositifs imitant ou remplaçant l'humain dans certaines mises en œuvre de ses fonctions cognitives"[1]

7.1 Flappy learning

Ce projet est une reprise du jeu *Flappy bird* qui est un jeu vidéo sorti sur smartphone en 2013. Dans ce jeu, vous contrôlez un oiseau devant naviguer entre des tuyaux et le but est d'aller le plus loin possible sans les toucher. Cette *IA* a été réalisée en *Javascript* sans aucune bibliothèque et utilise les principes de *Neuroevolution*. Tout son code est disponible sur *Github*[2]. Malheureusement, aucune documentation n'est présente, il n'y a que le code. La personne ayant créé cette *IA* se nomme apparemment Vincent BAZI et a un master en informatique.

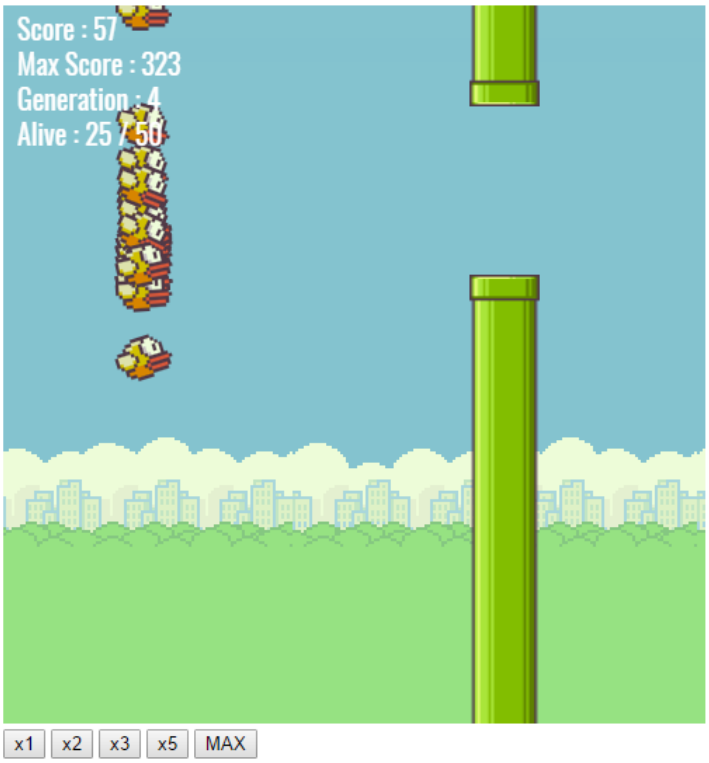


FIGURE 5 – Flappy Learning

La différence entre ce projet et le mien, mis à part le jeu, est que le réseau de neurones utilisé est beaucoup plus petit. *FlappyLearning* utilise un réseau de 3 neurones : 1 input, 1 hidden et 1 ouput. Le fait d'avoir un réseau de neurones plus petit ne change pas le code des réseaux de neurones mais plutôt le temps que l'*IA* va prendre à trouver comment avancer. Il implémente également le fait de pouvoir exécuter plusieurs oiseaux en parallèle pour ainsi obtenir un résultat plus rapidement. C'est une chose qui n'est pas prévue dans mon cahier des charges, mais que je souhaite implémenter (pas forcément sous cette forme). Ensuite, les principes du jeu *Flappy bird* son plus simple que le jeu du *T-Rex*. Là où le jeu du *T-Rex* possède plusieurs éléments variables tels que la vélocité, la taille d'un obstacle, sa position sur l'axe Y ou encore la distance par rapport à celui-ci, *Flappy bird* ne possède que la distance par rapport à un objet ainsi que sa position sur l'axe Y. Rendant ainsi le réseau de neurones bien moins complexe.

7.2 MarI/O

Ce projet est celui qui m'a donné envie d'utiliser le principe du *Machine learning* dans un jeu vidéo. Il reprend également un jeu vidéo qui est cette fois-ci *Super Mario World*, jeu sorti en 1990. Ce projet est certainement le plus gros projet fait par quelqu'un d'indépendant. Il a été créer par une personne connue sur *Youtube* se nommant *SethBling*. Les seules informations disponibles à son sujet sont qu'il a travaillé chez microsoft en tant que développeur sur *Xbox* et *Bing*.

Premièrement, la plus grosse différence avec tous les autres projets dont je vais parler est que tout cela a été fait sans avoir accès au code. Tous les autres projets cités ici ont tous un accès direct au code et aux valeurs souhaité, alors que MarI/O est une *IA* écrite en LUA pour un jeu qui n'est pas *open source* et qui est prévu pour fonctionner sur *Super Nintendo*.

Pour ce faire, cette personne a donc utilisé un emulateur (Bizhawk[8]) qui permet d'ajouter des scripts fait en LUA. Hormis le fait qu'il a dû développer un réseau de neurones énorme dû au nombre de variables et de bouton, il a dû synthétiser l'interface pour ne récupérer que ce qui est utile pour l'*IA*. Une autre des grosses différences est qu'il utilise un réseau de neurones de type NEAT [4] qui en plus d'altérer les paramètres "Weight" comme tous les types de réseau de neurones, altère également la structure du réseau, ce qui en fait un réseau de neurones très spécial et complexe. Ce projet est colossal et aura pris 2 mois au total (apprentissage de NEAT + développement).

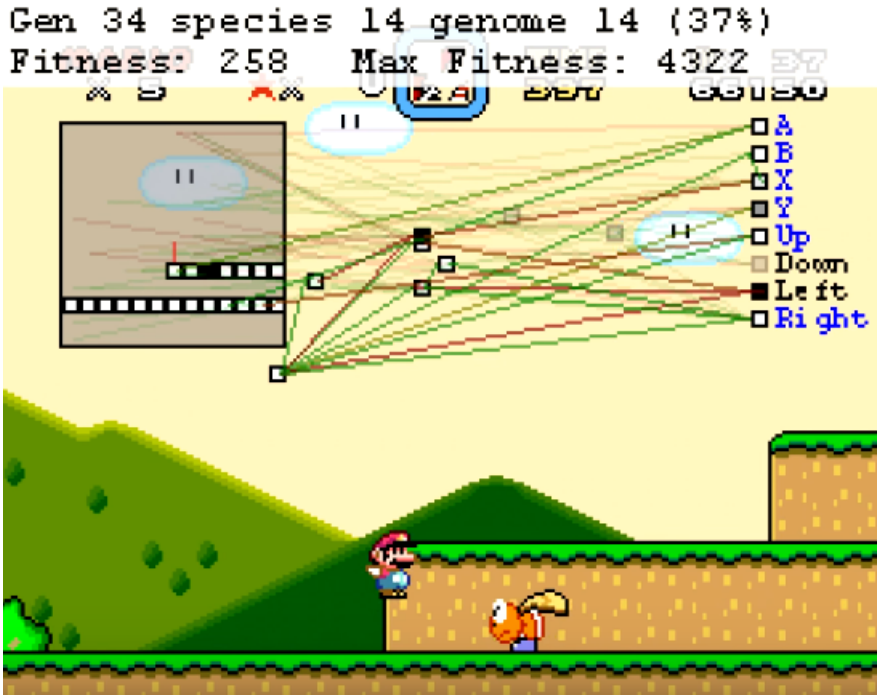


FIGURE 6 – MarI/O

Il est aussi important et intéressant de noter que ce projet aura pu être réutilisé (avec

4. NeuroEvolution of Augmenting Topologies[9]

quelque léger changement) pour fonctionner sur le jeu *Super Mario Kart* qui est un jeu complètement différent dans lequel on contrôle un kart, ainsi que le jeu *Super Mario Bros* dans lequel un *glitch*[5] a été trouvé grâce à celui-ci. Dû au fait que l'*IA* ait accès à plus de chose qu'un joueur (elle a accès à ce qui se passe réellement, pas ce qui est affiché à l'écran), l'*IA* s'est rendu compte qu'il était possible de passer par un passage normalement bloqué par un ennemi, lui faisant ainsi gagner du temps.

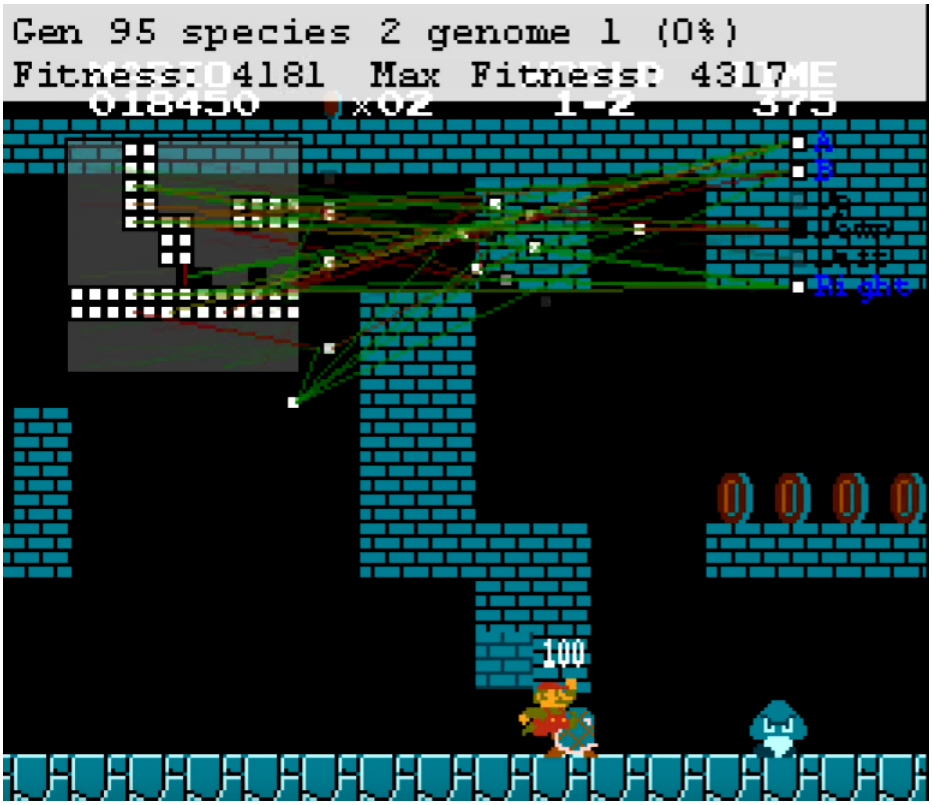


FIGURE 7 – Glitch trouvé dans *Super Mario Bros*

Malheureusement, encore une fois, aucune documentation n'est disponible, je ne pourrai donc pas en dire plus sur ce projet.

5. "terme employé pour désigner un bogue dans un jeu vidéo, où un objet animé a un comportement erroné (par exemple : passage au travers des murs, « téléportation » inattendue), qui peut-être exploité pour finir un jeu le plus vite possible"[10]

7.3 Forza Drivatar

Forza Drivatar est une *IA* utilisant le *Machine learning* qui a été créer par le studio de recherche de Microsoft au Cambridge dans le but d'être utilisé dans le jeu vidéo *Forza Motorsport*(jeu de course de voitures). Elle a été créée dans le but d'apprendre à se comporter comme un vrai joueur pour pouvoir le remplacer (une option est présente pour activer la conduite automatique) et éviter de rendre les *IA* trop prévisibles et similaires. Pour ce faire, elle récupère les informations de joueur humain et envoie ces informations dans une simulation.

Ce système fonctionnait, dans les 4 premiers opus, en local et donc calculé directement sur la console. Depuis *Forza Motorsport 5*, ce système est maintenant effectué depuis le *cloud* grâce au *Cloud Computing*[6], permettant ainsi d'augmenter grandement la puissance de calcul.

Ce projet est le seul que j'ai pu trouver qui a été développer "dans le jeu" et non pas en tant qu'ajout une fois celui-ci terminé. Malheureusement, aucune information sur le réseau de neurones n'a été donnée. Néanmoins, le réseau utilisé doit être semblable au réseau de neurones d'une voiture autonome et doit probablement être du *Reinforcement Learning*.

6. "exploitation de la puissance de calcul ou de stockage de serveurs informatiques distants"[11]

7.4 DeepMind et Starcraft

DeepMind est une entreprise ayant été racheté par Google qui est spécialisée dans l'*IA*. Son principale objectif est de développer une *IA* qui a pour objectif de résoudre même les problèmes les plus complexes sans pour autant lui dire comment faire. D'après eux, les jeux vidéo seraient le meilleur environnement pour faire cela, car ils leur permettraient de visualiser rapidement l'intelligence de l'*IA*.

DeepMind n'en est pas à ses premiers essais puisqu'ils sont les développeurs de *AlphaGo*. *AlphaGo* est la seule *IA* ayant réussi à battre un joueur professionnel (par ailleur, le champion du monde) de *Go*. Programmer un joueur de *Go* est quelque chose d'extrêmement complexe puisque, pour comparer à un jeu plus connu, le nombre de parties différentes possibles de *Go* est estimé à 10^{600}, là où les échecs en ont 10^{120}.

Pour revenir au sujet principal, *DeepMind* a choisi de développer une *IA* sur *StarCraft* (jeu de gestion de troupes), trouvant qu'il serait un bon environnement de développement dû au fait que les compétences requises pour jouer à *StarCraft* pourraient être utilisée pour effectuer des tâches dans la vrai vie.

Ce projet est le seul dont je parle qui utilise un réseau de neurones de type *Reinforcement Learning*(ou en tout cas le seul ou cela est affirmé). Ce type de réseau de neurones est utilisé lorsque l'on ne sait pas comment classifier les données, mais que nous savons quel résultat nous souhaitons. L'entraînement se fait avec des données fournies et requière un système de "récompense". Lorsque le résultat est correct on en informe l'algorithme en lui disant que ce qu'il a fait est correcte et lorsqu'il se trompe, on lui dit qu'il a mal fait quelque chose.

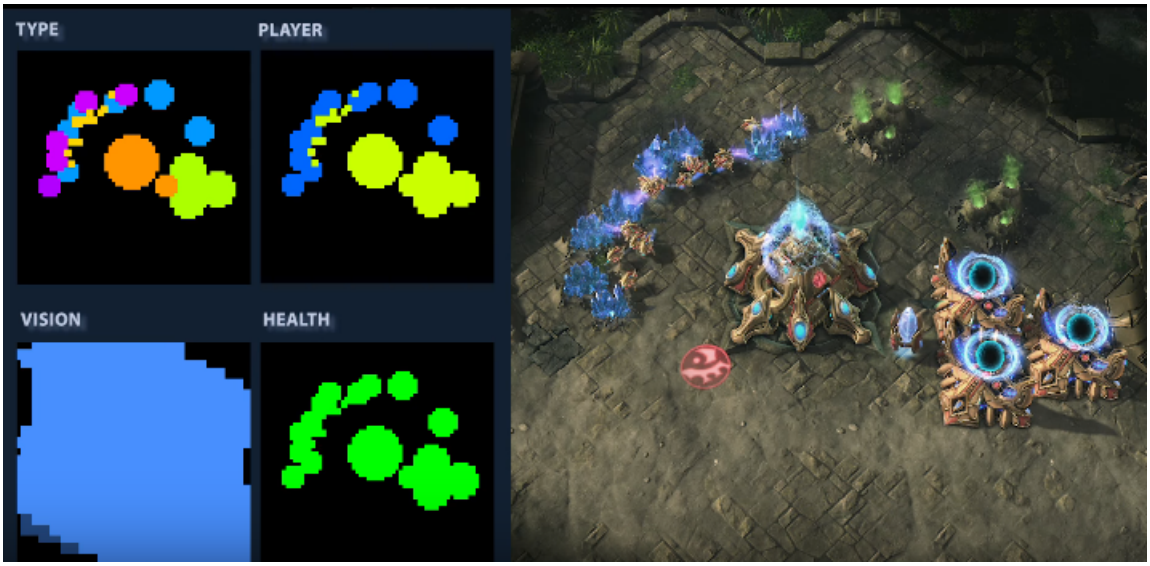


FIGURE 8 – À gauche, ce que voit l'*IA*. À droite, le jeu

8 Analyse fonctionnelle

8.1 Réseau de neurones

Le réseau de neurones est une des parties du programme qui permet de structurer toutes les données. Il agit comme un cerveau et imite le fonctionnement des neurones. Il se compose de plusieurs parties.

8.1.1 Modèle

Il faut commencer par, soit créer un modèle, soit en utiliser un existant. Le modèle définit la façon dont seront organisées nos informations et il impacte également les performances de l'application finale. Lors de ce travail, je vais donc créer un modèle moi-même. N'ayant jamais créé de modèle précédemment, je vais devoir me documenter et apprendre. Il faut savoir qu'il existe plusieurs types/catégories de modèle, les plus connus étant : *Supervised*, *Unsupervised* et *Reinforcement*.

— **Supervised** : Utilisé lorsque l'on sait comment classifier les données et que l'on souhaite juste les faire trier. Entraînées avec des données fournies et requièrent des données de test.

— **Unsupervised** : Utilisé lorsque l'on ne sait pas comment classifier les données. Requière des données de test, permettant au modèle de définir un "pattern" dans les données reçu.

— **Reinforcement** : Utilisé lorsque l'on ne sait pas comment classifier les données, mais que nous savons quel résultat nous souhaitons. Entraîné avec des données fournies et requière un système de "récompense". Lorsque le résultat est correct on en informe l'algorithme en lui disant que ce qu'il a fait est correcte et lorsqu'il se trompe, on lui dit qu'il a mal fait quelque chose.

Le modèle que je vais créer sera de type "Supervised". Ce n'est pas le modèle le plus optimisé pour ce genre de choses, mais utiliser un modèle *NEAT* sans avoir réalisé de réseau de neurones au préalable serait du suicide je pense, bien que je pensais l'utiliser au départ. J'utiliserai également les algorithmes génétiques dont je parlerais plus amplement plus tard. Voici un exemple d'un réseau de neurones pour le jeu du *T-Rex* (pas forcément celui qui sera utilisé).

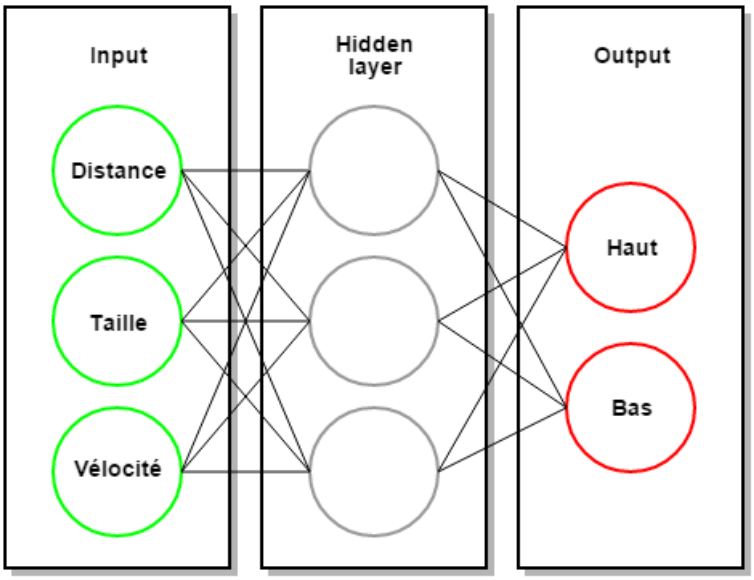


FIGURE 9 – Exemple du réseau de neurones du T-Rex

Un réseau de neurones se divise en trois parties :

— Input *layer*
— Hidden *layer*
— Output *layer*

8.1.2 Input

La partie "Input *layer*" représente toutes les caractéristiques qui influeront sur les actions du *T-Rex*. Dans ce jeu, plusieurs choses peuvent influencer le déclenchement du saut.

— **La distance :** permettant de connaître la distance séparant le T-Rex d'un obstacle
— **La taille de l'obstacle :** permettant de connaître la taille en largeur de l'obstacle le plus proche
— **La vélocité :** permettant de connaître la vitesse actuelle du T-Rex.
— **La position Y :** permettant de connaître la position sur l'axe Y de l'obstacle le plus proche.

Tous ces éléments sont importants au déclenchement du saut. Si le *T-Rex* ne connaît pas la distance le séparant de l'obstacle, il ne saura jamais quand sauté. De même si le *T-Rex* ne connaît ni sa vélocité ni la taille de l'obstacle, il ne comprendra jamais qu'il lui faut un timing précis pour un obstacle plus long ou bien avec une vitesse plus élevée. Les données d'entrées sont normalisées en utilisant les valeurs maximales possibles avant d'être envoyé dans d'autres neurones.

Étant donné que l'*IA* est développé par-dessus le jeu, elle a accès à toutes les valeurs citées précédemment. Il est néanmoins envisageable dans le futur de la créer en dehors de celui-ci et de récupérer ces informations uniquement en analysant l'interface graphique.

8.1.3 Hidden *layer*

Passons maintenant à la zone du milieu, la partie "Hidden *layer*". Cette partie ne contient qu'une seule chose, des neurones. Il est important de noter qu'il peut y avoir plusieurs "Hidden *layer*" mis côte à côte, cela complexifie le réseau de neurones et permet d'obtenir un résultat plus précis, mais cela prendra également plus de temps à l'exécution, il faut donc trouver le bon rapport complexité/temps. Ces neurones sont en fait simplement des fonctions que l'on appelle "fonction d'activation". Ils prennent en entrée deux nombres, qui sont le résultat du neurone précèdent ainsi qu'un poids. Ils transmettent ensuite le résultat aux neurones suivant et ainsi de suite jusqu'à arriver à la zone "output". Les poids sont représentés sur le schéma par des traits, il s'agit en fait simplement de nombres générés aléatoirement se trouvant dans un intervalle défini. chaque trait est un nombre différent.

8.1.4 Fonction d'activation

Pour revenir sur les fonctions d'activations, ce sont des fonctions définies qui permettent de normaliser les valeurs d'entrées dans une certaine porté, les plus connus étant : Sigmoid, ReLu, Tanh et Linear. Sigmoid permet par exemple de normaliser entre 0 et 1 alors que Tanh permet de normaliser entre -1 et 1. Il faut également savoir que, dans un réseau de neurones "standard", les seules valeurs qui peuvent varier sont les poids, ce sont les valeurs que le réseau de neurones va modifier jusqu'à obtenir le résultat escompté. Les valeurs que le réseau de neurones ne peut pas apprendre de lui-même sont appelées "Hyperparameter", ce type de paramètre doit être renseigné a la main (exemple : le nombre de neurones).

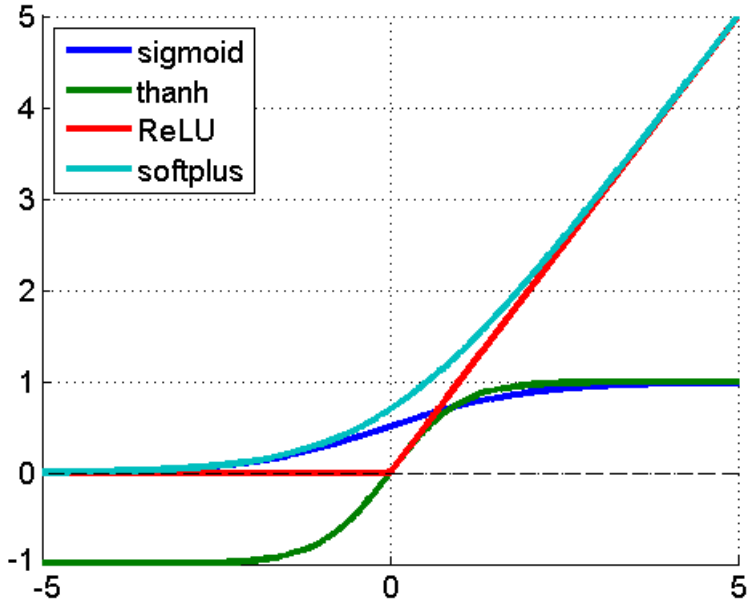


FIGURE 10 – Différente fonction d'activation

8.1.5 Output

Il ne reste maintenant plus que la troisième partie, la zone "Output *layer*". Cette zone est, comme son nom l'indique, la zone de sortie. Selon le résultat obtenu sur le neurone de sortie, une touche sera pressée ou non (dans le cas du *T-Rex*). Pour cela, il y aura simplement une fonction qui vérifiera le résultat obtenu, si le résultat est plus grand que 0.6 (par exemple), alors la touche est activée, sinon on ne fait rien. Il faut savoir que le résultat de sortie de chaque neurone est normalisé en général entre 0 et 1 (grâce aux fonction d'activation).

8.1.6 Entraînement

Dans le cas du jeu du *T-Rex* je ne pourrais pas utiliser les méthodes d'entraînement d'un réseau de type "supervised". Dans le jeu du *T-Rex* j'ai des données d'entrées (distance, etc..), mais pour faire du "supervised" il me faut également les valeurs de sortie, ce que je n'ai pas. Je ne peux pas lui dire "si tu as 10 pour la distance, 2 pour la vélocité, 9 pour la taille et 4 pour la position en Y, alors saute". Le but est qu'il apprenne de lui-même et en direct, pas que je lui dise "lis ces valeurs et trouve un pattern" pour ensuite le lancer sur le jeu pour qu'il ait un gros score.

À cause de cela je suis obligé de faire autrement, je vais donc mettre en place un algorithme génétique qui servira uniquement à l'entraînement de ce réseau.

8.2 *Algorithme génétique*

Les algorithmes génétiques me permettront de générer plusieurs réseaux de neurones qui sont appelés "Génome". En général, la seule variable d'un réseau de neurones sont ses poids, nous allons donc créer plein de réseau de neurones avec des poids différents dans le but de les faire se "reproduire" pour obtenir des "enfants" plus "forts".

Ces réseaux de neurones sont ensuite testés sur le jeu, puis, en faisant de la sélection naturelle, nous allons jeter ceux ayant le moins bon score. Ensuite, nous allons faire ce que l'on appelle de l'enjambement ("cross over" en anglais). Cela consiste à prendre des parties de chacun des réseaux de neurones restant pour les combiner en un seul. Par la suite, nous allons sélectionner aléatoirement des parties d'un génome auquel seront ajoutées des valeurs aléatoires (étape appelée "mutation"). Il ne reste plus qu'à répéter ce processus jusqu'à avoir le nombre de génome souhaité, créant ainsi une nouvelle génération. Tout ce processus (sélection, enjambement et mutation) est répété jusqu'à avoir le résultat escompté.

8.2.1 Sélection

La sélection est la première étape dans la mise en place d'un algorithme génétique. Elle consiste, comme son nom l'indique, à sélectionner les meilleurs réseaux de neurones. Pour cela il va falloir les évaluer. Dans mon cas, la valeur d'évaluation d'un réseau de neurones est le nombre d'obstacles que l'*IA* a réussis à passer. Cette valeur est appelé "fitness". il

existe plusieurs noms différents selon le type de résultat que l'on souhaite avoir. Si nous avons une *IA* qui a pour objectif d'obtenir un score le plus petit possible (réduire un maximum les dégâts / minimiser quelque chose) alors on utilisera le terme "cost". Si, a l'inverse, le but est d'obtenir le plus gros score possible, alors le terme "fitness" sera utilisé. Le principe de la sélection est simple, selon le nombre de génomes par génération, le nombre de parents sélectionnés change. Plus il y a de génomes, plus il y aura de parents sélectionnés. Cela permet d'augmenter la diversité et ainsi d'explorer plus de solutions. Une fois la sélection terminée, on envoie tous les parents pour qu'ils se reproduisent.

8.2.2 Enjambement

Cette méthode consiste à créer des enfants basés sur deux parents qui ont été sélectionnés. Les enfants sont créés en mélangeant les gènes de chacun des deux parents. Son but est de produire des enfants différents les uns des autres pour ainsi explorer plus de possibilités.

Il existe plusieurs méthodes d'enjambement, je parlerais ici de la méthode en un point (one point crossover) qui est une des plus simples à comprendre et à expliquer. Pour cela, on commence par tirer un point aléatoirement dans un des deux parents et on divise en deux les deux parents à partir de ce point. Il ne reste plus qu'à prendre la 1er partie du premier parent et de la mettre avec la 2ème partie du deuxième parent ainsi que la 1er partie du deuxième parent avec la 2ème partie du premier parent. Grâce à ça, on obtient deux enfants qui sont différent de leurs parents. Les autres méthodes consistent simplement à prendre plus de point et diviser les parents en plus de partie.

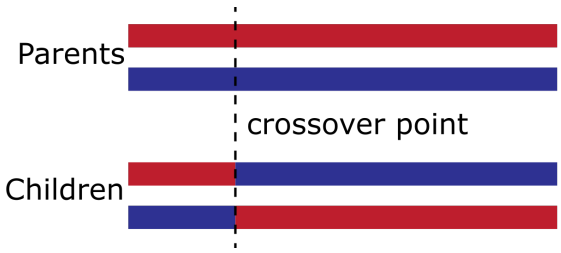


FIGURE 11 – Méthode d'enjambement "crossover" en un point

8.2.3 Mutation

La dernière étape d'un algorithme génétique consiste à faire muter certaines parties des gènes des enfants. Son but est d'empêcher d'être bloqué dans des optima locaux. C'est une étape très importante puisque si deux parents sont identiques, les enfants créés le seront également. La mutation permet donc de sortir de cette boucle infinie.

Si on prend une courbe qui a plusieurs piques et que notre but est de trouver quel est le point culminant, en utilisant des algorithmes génétiques ou d'autres méthodes, il est possible que l'algorithme trouve un point qu'il considère comme étant le plus élevé et va donc s'arrêter de chercher alors qu'il existe un point plus loin qui est plus élevé, c'est ce que l'on appelle un optimum local. Le but de la mutation va donc être de dire à chaque enfant qu'il faut regarder plus loin en les modifiant légèrement pour peut-être trouver un optimum global.

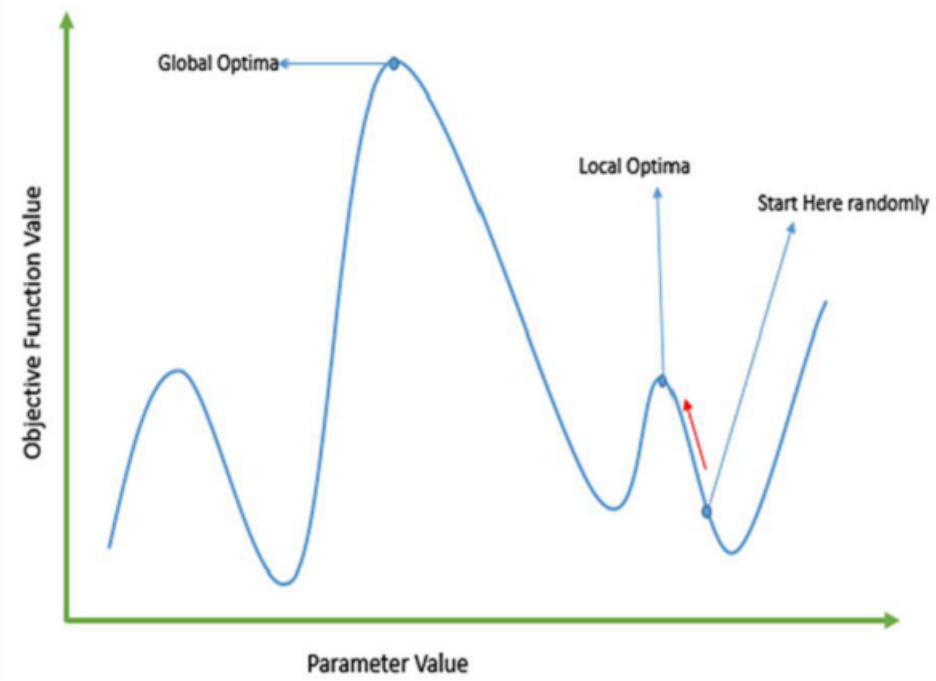


FIGURE 12 – Exemple d'optimum global et d'optimum local

Il existe plusieurs méthodes de mutation et l'utilisation de chacune dépend du besoin ou des valeurs dans les gènes des parents (binaire, nombre, valeur dans une portée donnée, etc..). Personnellement j'ai choisi de faire de la mutation avec un taux. Les gènes de chaque enfant sont parcouru et chaque partie des gènes à X% de chance d'être modifié. Pour ce qui est de la modification des gènes, une valeur entre -1 et 1 est tirée aléatoirement et ajoutée à l'ancienne valeur du gène.

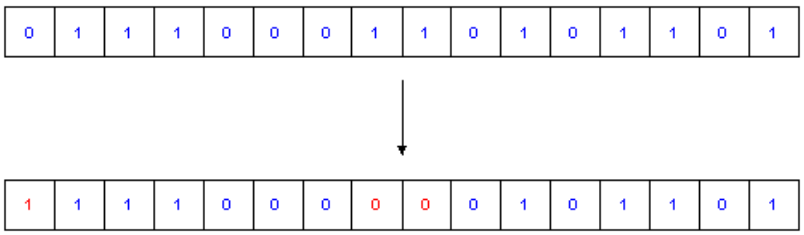


FIGURE 13 – Exemple d'une méthode de mutation

8.3 Maquette de l'interface

Cette application contient une seule page se divisant en plusieurs parties.

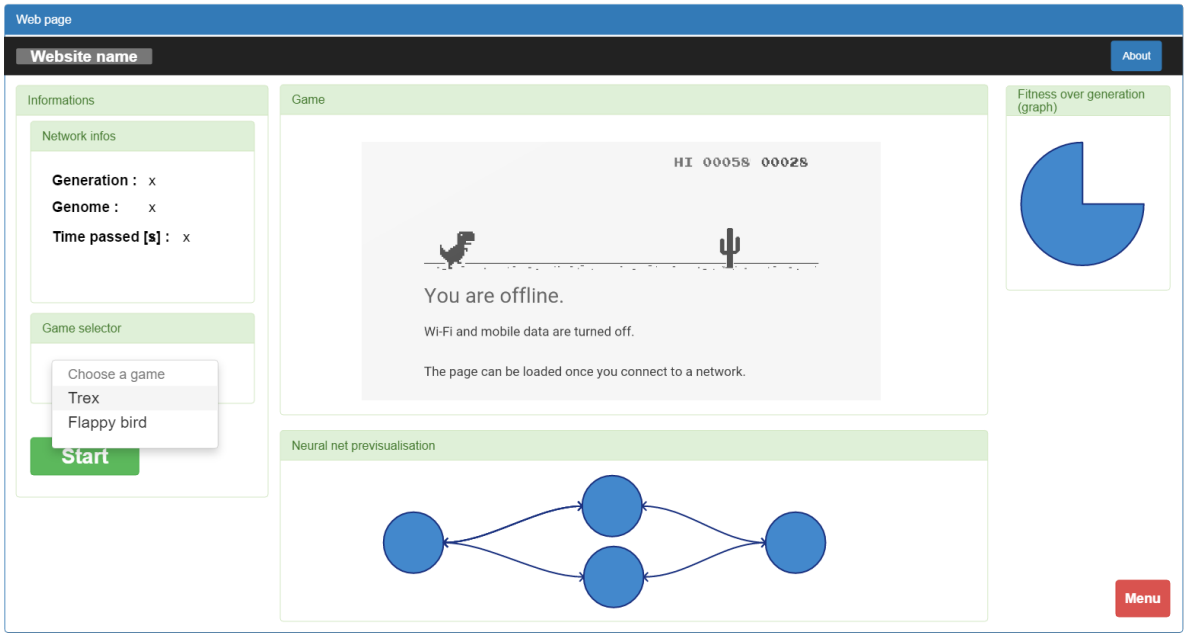


FIGURE 14 – Page principale

8.3.1 Partie gauche

Tout d'abord, la partie "Network Infos" qui permet d'afficher plusieurs informations sur le réseau de neurones actuel. "generation" permet d'afficher l'index de la génération sur laquelle est actuellement l'*IA*. "genome" qui affiche l'index du génome sur lequel est actuellement l'*IA* et pour finir "Time passed" qui permet simplement d'afficher le temps en secondes pendant lequel l'*IA* a fonctionné.

En dessous se trouve la partie "Game selector" qui contient uniquement un selector permettant de changer de jeu.

Pour finir, un bouton "start" ou "stop" est présent pour pouvoir démarrer ou arrêter l'*IA*.

8.3.2 Milieu

La partie du milieu contient le jeu et permet d'afficher en direct les actions de l'*IA* pour avoir un retour et voir les résultats de son entraînement.

En dessous se trouve le réseau de neurones sous forme graphique. Cela permet de voir comment le réseau va traiter les données et quelles sont les valeurs.

8.3.3 Partie droite

Cette partie contient uniquement un graphique qui permet de voir le score des réseaux de neurones sur chaque génération, cela permet d'avoir un suivi et de voir à quel point il apprend vite ou non.

8.3.4 Menu

Un menu s'ouvre lorsque l'utilisateur clique sur le bouton présent en bas à droite de la fenêtre.

Ce menu permet d'accéder à deux fenêtres. La première étant la fenêtre d'exportation qui permet d'exporter un réseau de neurones. La seconde, la fenêtre d'importation, qui permet d'importer un réseau de neurones.

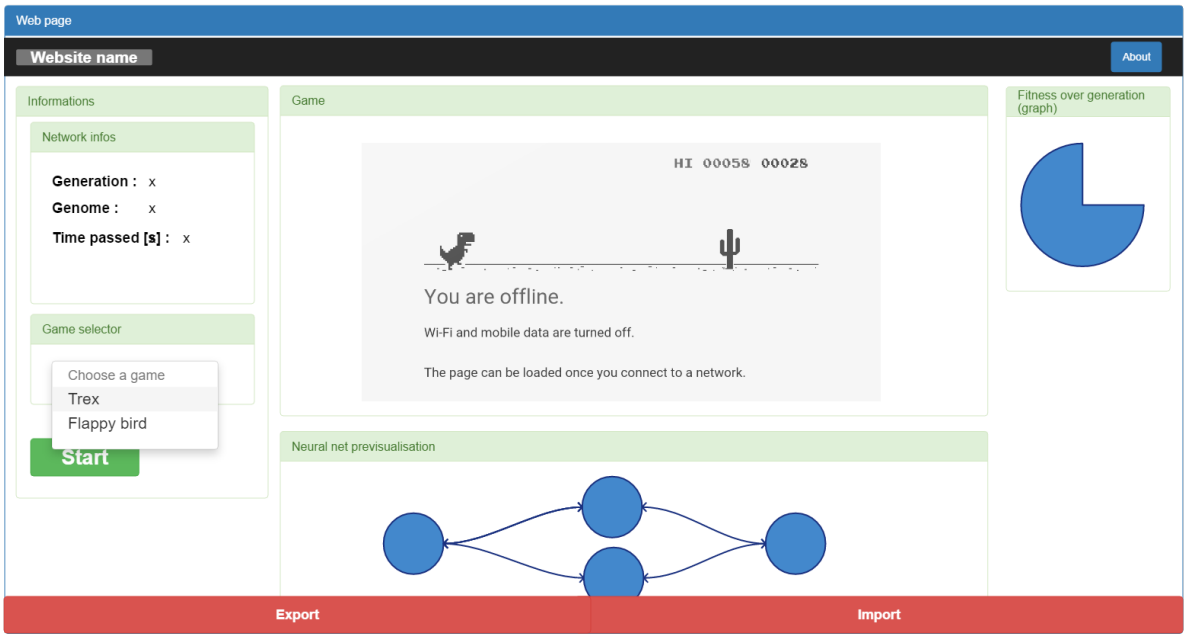


FIGURE 15 – Barre de menu

8.3.5 Fenêtre d'exportation

Cette fenêtre permet, comme son nom l'indique, d'exporter un réseau de neurones. Pour cela il y a plusieurs possibilités. Il est possible de sélectionner à la main le réseau que l'on souhaite exporter en sélectionnant son index dans une liste déroulante. La seconde option est de cliquer sur le bouton "Export the best neural net" qui permet d'exporter le meilleur réseau de neurones. Ces deux actions vont afficher les valeurs du réseau sélectionné dans la zone de texte présent en dessous, il sera alors possible de simplement copier-coller ces informations dans la fenêtre d'importation, ou bien il est possible de télécharger le réseau de neurones sous format texte en cliquant sur le bouton "Export to file".

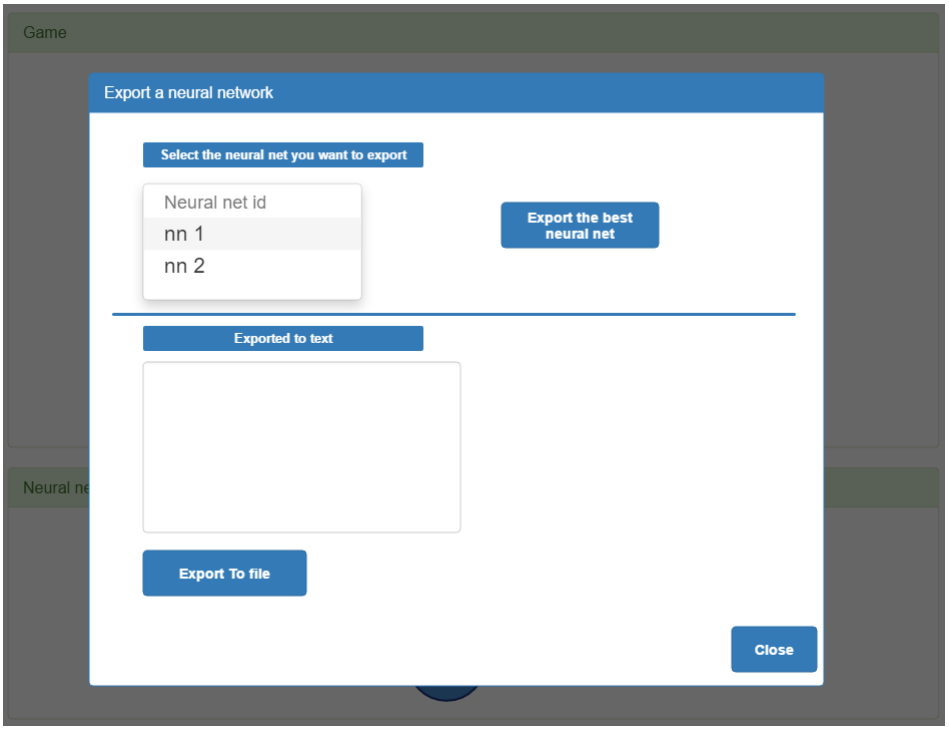


FIGURE 16 – Fenêtre d'exportation

8.3.6 Fenêtre d'importation

Cette fenêtre permet d'importer un réseau de neurones précédemment exporté, pour cela il y a plusieurs possibilités. La première est de sélectionner un fichier exporté contenant un réseau de neurones. La seconde est de copier-coller le texte dans la zone présente en dessous. Lors de l'importation, le texte présent dans le fichier sera affiché dans la zone en dessous. Il ne restera plus qu'à cliquer sur le bouton "Import neural network" pour l'importer. Si le réseau ne correspond pas et qu'il est impossible de l'importer (mauvais jeu, manque d'information, etc..) une erreur sera alors affichée pour prévenir l'utilisateur.

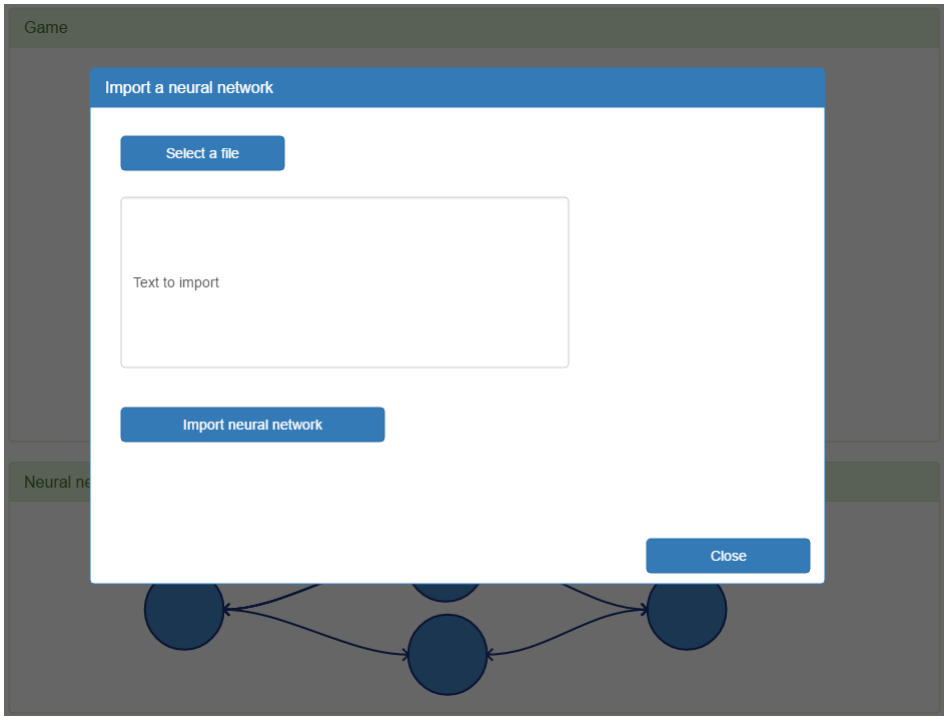


FIGURE 17 – Fenêtre d'importation

8.3.7 Fenêtre "about"

Cette fenêtre permet simplement d'expliquer le but et les possibilités de ce projet.

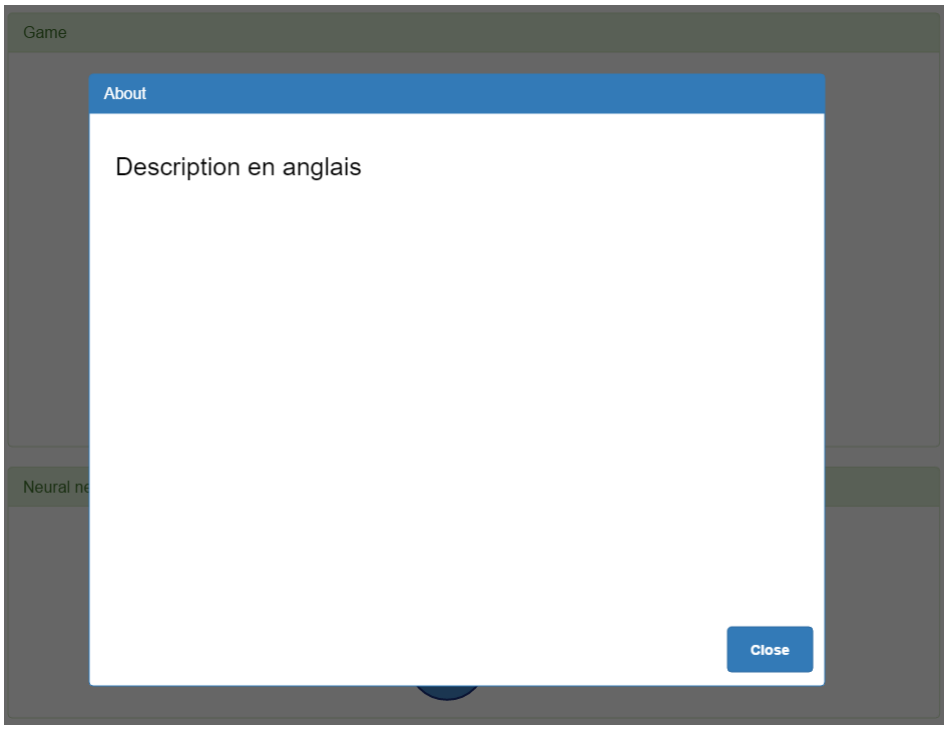


FIGURE 18 – Fenêtre "A propos"

8.4 Carte de navigation du site

Voici la carte de navigation du site. La fenêtre "about" et le menu ne sont accessibles que depuis la page principale (home), alors que les pop-ups "Export" et "Import" sont seulement accessibles depuis le menu.

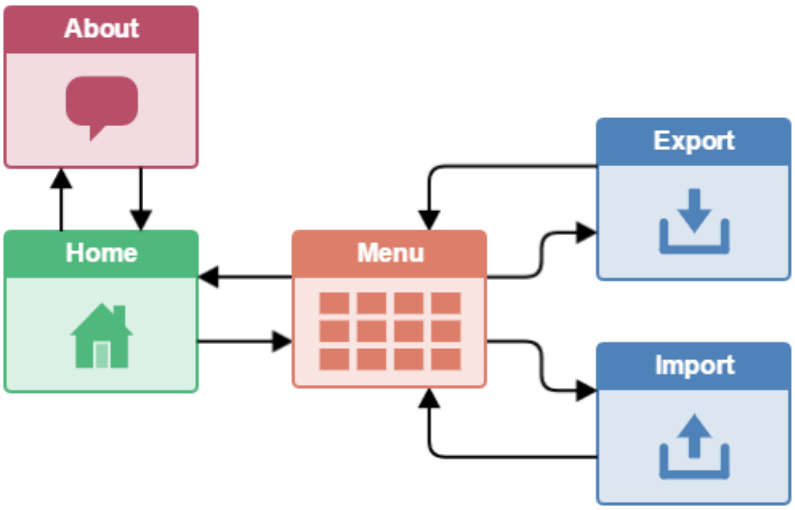


FIGURE 19 – Carte de navigation

9 Analyse organique

9.1 Diagramme de classe

Voici le diagramme de classes simplifié. Malheureusement, étant donné qu'il est trop grand, les méthodes ainsi que les champs ont été retirés. Les diagrammes détaillés se trouvent dans les classes respectives.

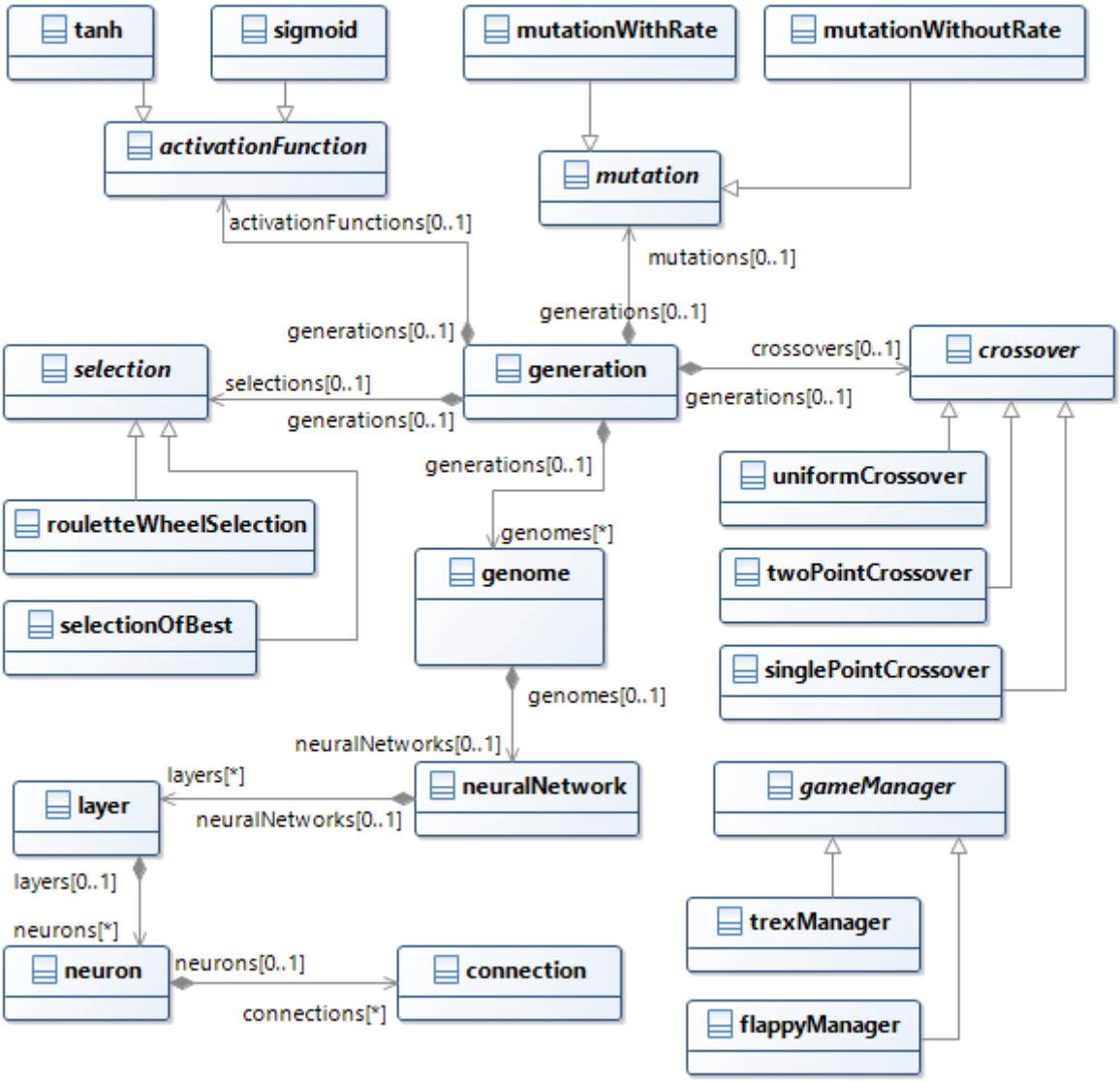


FIGURE 20 – Uml simplifié

9.2 Main

Ce fichier est le fichier principal de l'application. Il s'occupe d'instancier une génération et de gérer tous les éléments de l'interface. Il se divise en plusieurs parties.

La première partie est la partie "document.ready()", c'est cette partie qui va tout instancier et elle se lance uniquement lorsque le site a fini de faire tous les chargements. Les deux premiers gros bloc (modal et data) permettent respectivement de configurer les modals et le graphique. Le bloc d'après (chartCtx) configure également le graphique. Les trois lignes suivantes servent à activer plusieurs fonctionnalités graphiques de la bibliothèque "materiallizecss".

C'est seulement après tout cela que commence vraiment le code. les lignes 69 à 72 permettent de stocker certains paramètres du réseau de neurones. Elles seront utilisées lors de son instanciation. Les lignes 75 à 84 permettent d'instancier le bon gameManager (selon une variable globale) ainsi que changer le selector en fonction. Ensuite, le jeu est instancié ainsi qu'une génération. Le code en dessous permet de changer le jeu en direct et de mettre à jour quelques éléments graphiques. Pour finir, les 3 dernières lignes permettent de lancer l'*IA*.

```
1  $(document).ready(function(){
2    // Enable the modals and configure them
3    $('.modal').modal({
4      dismissible: true, // Modal can be dismissed by clicking outside ↩
     of the modal
5      opacity: .5, // Opacity of modal background
6      inDuration: 300, // Transition in duration
7      outDuration: 200, // Transition out duration
8      startingTop: '4%', // Starting top style attribute
9      endingTop: '10%', // Ending top style attribute
10     ready: function(modal, trigger) { // Callback for Modal open. ↩
     Modal and trigger parameters available.
11       //console.log(modal, trigger);
12     },
13     complete: function() { // Callback for Modal close
14       //Clear the modals on close
15       clearModals();
16     }
17   });
18
19   // Configuration of the chart
20   var data = {
21     datasets: [
22       {
23         label: "Best fitness",
24         fill: true,
25         lineTension: 0.1,
26         backgroundColor: "rgba(75,192,192,0.4)",
27         borderColor: "rgba(75,192,192,1)",
28         borderCapStyle: 'butt',
29         borderDash: [],
30         borderDashOffset: 0.0,
31         borderJoinStyle: 'miter',
32         pointBorderColor: "rgba(75,192,192,1)",
33         pointBackgroundColor: "#fff",
34         pointBorderWidth: 1,
35         pointHoverRadius: 5,
36         pointHoverBackgroundColor: "rgba(75,192,192,1)",
37         pointHoverBorderColor: "rgba(220,220,220,1)",
```

```
38          pointHoverBorderWidth: 2,
39          pointRadius: 5,
40          pointHitRadius: 10,
41          spanGaps: false,
42        }
43      ]
44    };
45
46    //Creation of the empty chart
47    var chartCtx = document.getElementById("Chart");
48    fitnessChart = new Chart(chartCtx, {
49      type: 'line',
50      data: data,
51      options: {
52        title: {
53          display: true,
54          text: 'Best fitness over generations' // Title
55        }
56      }
57    });
58
59    // Event when a file is loaded
60    document.getElementById('selectedFile').addEventListener('change', ↩
       readSingleFile, false);
61
62    // Call "resizeNeuralNetCanvas" when the window has been resized
63    window.addEventListener('resize', resizeNeuralNetCanvas, false);
64
65    // Enable the selector
66    $(document).ready(function() { $('select').material_select(); });
67
68    // Neural net configuration
69    activation = new sigmoid();
70    selectionMethod = new rouletteWheelSelection();
71    crossoverMethod = new singlePointCrossover();
72    mutationMethod = new mutationWithRate(0.2); // [0..1]
73
74    // Check wich game should be instanciate
75    if (currentGameIndex == games.TREX) {
76      $("select[name=gameSelector] option[value=1]").prop('selected', ↩
       'selected');
77
78      myGameManager = new trexManager();
79    }
80    else if (currentGameIndex == games.FLAPPY) {
81      $("select[name=gameSelector] option[value=2]").prop('selected', ↩
       'selected');
82
83      myGameManager = new flappyManager();
84    }
85
86    // Instanciate the game
87    runner = myGameManager.instanciateGame();
88
89    // Create a new generation
90    gen = new Generation(myGameManager.defaultTopology, ↩
       myGameManager.defaultNumberOfGenomes, activation, selectionMethod, ↩
       crossoverMethod, mutationMethod);
91
92    // Update the interface
93    updateInterface();
94
95    // Resize the canvas depending the container size
96    resizeNeuralNetCanvas();
```

```
97
98    // Print the value of the neural net on the "export" modals
99    $( "#neuralNetSelector" ).change(function() {
100     printSelectedNeuralNet($('#neuralNetSelector').val());
101   });
102
103   // Event when the game is changed
104   $( "select[name=gameSelector]" ).change(function() {
105     // Check if the selected option is the same as before
106     if (currentGameIndex != $('select[name=gameSelector]').val()) {
107       currentGameIndex = $('select[name=gameSelector]').val();
108
109       // Change the game depending the option selected
110       if (currentGameIndex == games.TREX) {
111         $('.interstitial-wrapper').html('<div id="main-content"><div ↩
              class="icon icon-offline" alt="" style="visibility: ↩
              hidden;"></div></div>');
112         $('.interstitial-wrapper').append('<div ↩
              id="offline-resources"><img id="offline-resources-1x" ↩
              src="assets/default_100_percent/100-offline-sprite.png"> ↩
              <img id="offline-resources-2x" ↩
              src="assets/default_200_percent/200-offline-sprite.png"> ↩
              </div>');
113         myGameManager = new trexManager();
114       }
115       else if (currentGameIndex == games.FLAPPY) {
116         myGameManager = new flappyManager();
117       }
118
119       // Instanciate the game
120       runner = myGameManager.instanciateGame();
121
122       //Create a new generation
123       gen = new Generation(myGameManager.defaultTopology, ↩
              myGameManager.defaultNumberOfGenomes, activation, ↩
              selectionMethod, crossoverMethod, mutationMethod);
124
125       // Draw the neural net
126       gen.drawNeuralNet(c,ctx);
127
128       // Reset the chart
129       fitnessChart.data.datasets[0].data = null;
130
131       //Change the running state
132       changeRunningState(false);
133     }
134   });
135
136   // Timer for the application
137   setInterval(function(){
138     startAI();
139   }, 1000 / AIResfreshRate);
140 });
```

9.2.1 changeRunningState

Cette fonction permet de changer l'état de l'*IA*, soit selon l'état précédent, soit selon le paramètre donné. Elle affiche également un message à l'utilisateur pour l'informer de l'état de l'*IA*.

```
1  // Change the state of the AI
2  function changeRunningState(state) {
3    if (typeof state == "undefined") {
4      isRunning = !isRunning;
5    }
6    else {
7      isRunning = state;
8    }
9
10   // Show if the AI is started or stopped
11   if (isRunning == true) {
12     $("#stateBtn").html('STOP<i class="material-icons ↩
         right">power_settings_new</i>');
13     Materialize.toast('AI Started', 4000);
14
15     // Unpause the game
16     myGameManager.play(runner);
17   }
18   else{
19     $("#stateBtn").html('START<i class="material-icons ↩
         right">power_settings_new</i>');
20     Materialize.toast('AI Stopped', 4000);
21
22     // Pause the game
23     myGameManager.pause(runner);
24   }
25 }
```

9.2.2 startAI

Cette fonction est celle qui va être exécutée X fois par seconde. Premièrement, elle regarde si l'*IA* devrait fonctionner. Si c'est le cas, elle va récupérer les valeurs du jeu actuellement affichées et si ces valeurs existes (permet de savoir si le jeu a toutes les valeurs requises à l'*IA*) alors on va incrémenter le temps passé, rafraîchir l'aperçu du réseau de neurones et de lui envoyer ces données. Une fois que ces données ont été envoyer, on regarde le résultat de sortie du réseau de neurones et on effectue une action en conséquence (ex : sauter). pendant ce temps, on met à jour le fitness (score) de l'*IA* et on regarde si elle est morte. Si c'est le cas, on redémarre le jeu, change de génération et met à jour l'interface.

```
1  function startAI() {
2    // Check if the AI should "run"
3    if (isRunning == true) {
4      // Get the normalized data
5      var gameValue = myGameManager.getNormalizedInputValues(runner);
6
7      // Check if some value were found
8      if (gameValue.length > 0) {
9
10       // timer
11       timer++;
12       if (timer >= AIResfreshRate) {
```

```
13          timePassed++;
14          $("#timeIndex").html(timePassed);
15          timer = 0;
16        }
17
18        // Neural net preview refresh
19        nnPreviewCounter++;
20        if (nnPreviewCounter >= (AIResfreshRate / ↩
              nnPreviewRefreshRate)) {
21          // Draw the neural net
22          gen.drawNeuralNet(c,ctx);
23          nnPreviewCounter = 0;
24        }
25
26        // Run the generation
27        var result = gen.run(gameValue);
28
29        // Make a game action
30        myGameManager.action(runner, result);
31
32        // Update the fitness of the AI
33        myGameManager.fitness(runner);
34
35        // When the AI die
36        if (myGameManager.isDead(runner)) {
37          // Restart the game
38          myGameManager.restart(runner);
39
40          // Change the generation and save the fitness
41          gen.nextGen(myGameManager.tmpFitness, fitnessChart);
42
43          // Reset the value (counting obstacle)
44          fitness = 0;
45          lastValue = 1000
46
47          // Update the interface
48          updateInterface();
49        }
50      }
51    }
52  }
```

9.2.3 getBestNN

Cette fonction permet simplement de récupérer le meilleur réseau de neurones.

```
1  // Get the best neural net index
2  function getBestNN(){
3    var maxFitness = -1;
4    var bestNNIndex;
5
6    for (var i = 0; i < gen.genomes.length; i++) {
7      if (gen.genomes[i].fitness > maxFitness) {
8        maxFitness = gen.genomes[i].fitness;
9        bestNNIndex = i;
10     }
11   }
12
13   printSelectedNeuralNet(gen.genomes[bestNNIndex]);
14 }
```

9.2.4 printSelectedNeuralNet

Cette fonction permet d'afficher les poids d'un réseau de neurones dans une "textarea".

```
1 // Print the weights of a neural net
2 function printSelectedNeuralNet(bestGenome) {
3   var weights = bestGenome.getWeights();
4   $("#textareaExport").html(weights.toString());
5 }
```

9.2.5 resfreshNNselection

Cette fonction permet de changer les valeurs du "selector" sur la fenêtre "export".

```
1  // Refresh the neural net index selector (export modals)
2  function resfreshNNselection() {
3    $("#neuralNetSelector").empty();
4    for (var i = 0; i < gen.genomes.length; i++) {
5      $("#neuralNetSelector").append('<option value="'+i+'">Neural net ↩
         '+(i+1)+'</option>');
6    }
7
8    $('select').material_select();
9
10   var nnIndex = $('#neuralNetSelector').val();
11   printSelectedNeuralNet(gen.genomes[nnIndex]);
12 }
```

9.2.6 updateInterface

Cette fonction permet de mettre à jour certaines valeurs de l'interface.

```
1 // Update the interface
2 function updateInterface() {
3   $('#generationIndex').html(gen.generation + 1);
4   $('#genomeIndex').html(gen.currentGenome + 1 + "/" + ↩
      gen.genomes.length);
5 }
```

9.2.7 resizeNeuralNetCanvas

Cette fonction permet de changer la taille de l'aperçu du réseau de neurones selon la taille de la fenêtre.

```
1 // Resize the canvas depending de windows size
2 function resizeNeuralNetCanvas() {
3   neuralNetCanvas.width = $("#neuralNetPreview").width();
4   neuralNetCanvas.height = $("#neuralNetPreview").height();
5
6   // Redraw the neural net after changing the size
7   gen.drawNeuralNet(c,ctx);
8 }
```

9.2.8 addDataToChart

Cette fonction permet d'ajouter une valeur à une "chart".

```
1 // Add a fitness to a chart
2 function addDataToChart(chart,fitness) {
3   var chartLength = chart.data.datasets[0].data.length;
4
5   chart.data.datasets[0].data.push(fitness);
6   chart.data.labels.push("Gen " + (chartLength + 1));
7   chart.update();
8 }
```

9.2.9 simulateKeyPress

Cette fonction permet de simuler la pression d'une touche sur le document.

```
1 // Simulate a key press
2 // 38=up, 40=down, 32=space
3 // "keyup", "keydown"
4 function simulateKeyPress(keycode, type) {
5   var evt = new Event(type);
6   evt.keyCode=keycode;
7   evt.which=evt.keyCode;
8   document.dispatchEvent(evt);
9 }
```

9.2.10 downloadFile

Cette fonction permet de télécharger un fichier en format texte. Elle prend en paramètre un nom de fichier et du texte.

```
1  // Download the neural network (exported neural network)
2  function downloadFile(filename, text) {
3    var element = document.createElement('a');
4    element.setAttribute('href', 'data:text/plain;charset=utf-8,' + ↩
       encodeURIComponent(text));
5    element.setAttribute('download', filename);
6
7    element.style.display = 'none';
8    document.body.appendChild(element);
9    element.click();
10   document.body.removeChild(element);
11
12   Materialize.toast('Neural network succesfully exported', 4000);
13 }
```

9.2.11 readSingleFile

Cette fonction permet de charger un fichier et d'importer le texte dans une "textearea".

```
// Read a file and print it in a textarea
function readSingleFile(evt) {
  //Retrieve the first (and only!) File from the FileList object
  var file = evt.target.files[0];

  // If the file exist/is selected
  if (file) {
    var reader = new FileReader();

    // Load the file
    reader.onload = function(e) {
      // Get the file data
      var contents = e.target.result;

      // Add the content to the textarea
      $("#textareaImport").val(contents);
    }

    // Read the text file
    reader.readAsText(file);
  } else {
    Materialize.toast('Failed to load file', 4000);
  }
}
```

9.2.12 clearModals

Cette fonction permet de vider les champs des fenêtres lorsqu'elles sont fermées.

```
// Clear the modals
function clearModals() {
  $('.modal').find('input:text, input:password').val('');
  $('.modal').find('input:radio, input:checkbox').prop('checked', ↩
   false);
}
```

9.2.13 importNeuralNetwork

Cette fonction permet d'importer un réseau de neurones. Pour cela, on regarde si un génome de la génération actuelle est de même taille que celui qui doit être importé, si ce n'est pas le cas, alors il est impossible de l'importer et l'utilisateur en est informé.

```
// Import a neural network
function importNeuralNetwork(){
  gen = new Generation(myGameManager.defaultTopology, ↩
      myGameManager.defaultNumberOfGenomes, activation, ↩
      selectionMethod, crossoverMethod, mutationMethod);

  var genToImport = JSON.parse("[" + $("#textareaImport").val() + "]");
  var oldGen = gen.genomes[0].getWeights();

  if (genToImport.length != oldGen.length) {
```

```
9       Materialize.toast('Importation error, missing or too many data ↩
            (wrong game ?)', 6000)
10    }
11    else {
12      gen.genomes[0].setWeights(genToImport);
13
14      Materialize.toast('Neural network succesfully imported', 4000)
15
16      // Reset everything
17      myGameManager.restart(runner);
18      removeData(fitnessChart);
19      timePassed = 0;
20      $("#timeIndex").html(timePassed);
21      updateInterface()
22      gen.drawNeuralNet(c,ctx);
23    }
24
25    $('#importModal').modal('close');
26  }
```

9.2.14 removeData

Cette fonction permet de vider une "chart" de toutes ses données.

```
1  // Remove all the data of a chart
2  function removeData(chart) {
3    chart.data.datasets[0].data = null;
4    chart.data.labels = [];
5    chart.update();
6  }
```

9.2.15 ravel

Cette fonction permet de transformer un tableau 2d en tableau 1d.

```
1  // Transform a multidimensionnal array into an 1 dimension array
2  function ravel(array) {
3    var result = new Array();
4    if (typeof array[0] == "undefined" || typeof array[0] == "number") {
5      result = array;
6    }
7    else {
8      for (var i = 0; i < array.length; i++) {
9        for (var j = 0; j < array[i].length; j++) {
10         result.push(array[i][j]);
11       }
12     }
13   }
14
15   return result;
16 }
```

9.3 Flappy

Cette classe fait partie d'un projet que j'ai récupéré sur internet. quelque parties ont dû être modifié pour la rendre compatible avec mon projet. Un objet a été créé pour englober le jeu et ainsi me permettre d'effectuer des manipulations depuis mon code. Mis à part cela, beaucoup de lignes ont été retiré et un statut a été ajouté pour me permettre de mettre le jeu en pause.

```
1  class flappyBird{
2    constructor(){
3      this.sprites = {
4        bird: document.getElementById("flapBird"),
5        background: document.getElementById("flapBackground"),
6        pipetop: document.getElementById("flapPipeTop"),
7        pipebottom: document.getElementById("flapPipeBottom")
8      }
9
10     this.game;
11     this.start();
12   }
13
14   start(){
15     this.game = new Game();
16     this.game.start();
17     this.game.update();
18     this.game.display();
19   }
20 }
```

9.4 Trex

Cette classe a été récupéré sur internet (une personne a exporté ce jeu de chromium) et a été légèrement modifié. Premièrement, l'instanciation automatique du jeu a été retiré. Ensuite, certaines fonctions ont été retiré car jugé inutile ou contraignante (ex : fonction mettant en pause le jeu lorsque la fenêtre n'a pas le focus). Pendant le projet, une version permettant d'instancier plusieurs *T-Rex* en même temps a été développé. Le problème est que le jeu ne le permet pas, trop de variables globales sont modifiées par les *T-Rex* (ex : la gravité lors d'un saut). Cette solution a donc été abandonnée.

9.5 GameManager

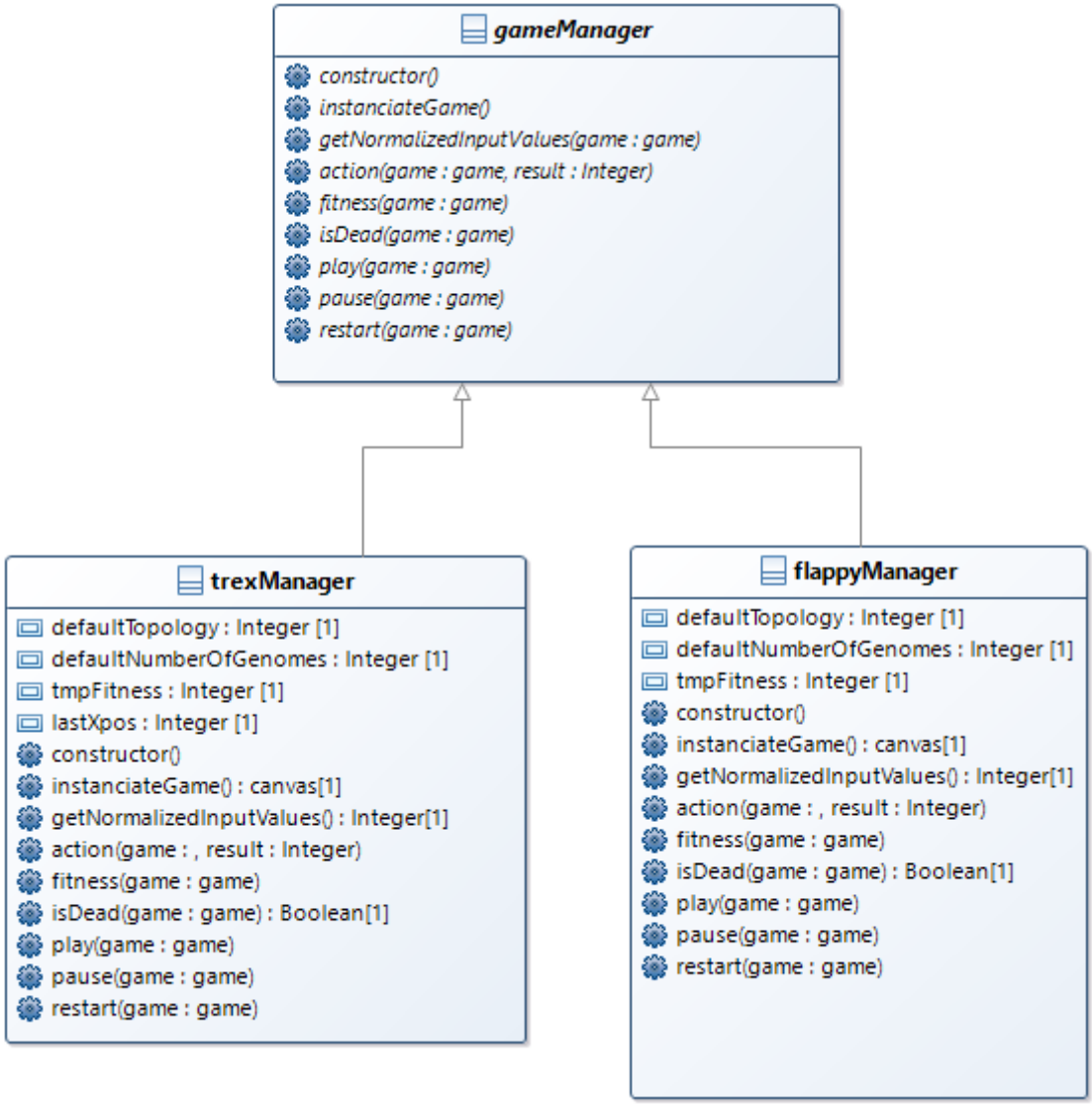


FIGURE 21 – Uml gameManager

Classe abstraite permettant d'avoir un modèle pour les différents jeux présents ou non dans l'application. Elle contient un constructeur qui empêche son instanciation, ainsi que 8 méthodes permettant toutes de, soit effectuer une action dans le jeu, soit de trouver une information. Le but de cette classe est de simplifier l'ajout de jeu. Les seules choses à faire lors de l'ajout d'un jeu sont uniquement de rajouter une classe qui héritera de gameManager. Il faudra également ajouter quelques lignes à d'autre endroits (ajouter le choix dans le "selector").

```
1 /** Abstract class for game manager */
2 class gameManager {
3   constructor() {
4     if (new.target === gameManager) {
5       throw new TypeError("Cannot construct gameManager instances ↩
          directly");
```

```
    }
  }

  instanciateGame() { throw new Error("Must override method"); }
  getNormalizedInputValues(game) { throw new Error("Must override method"); }
  action(game, result) { throw new Error("Must override method"); }
  fitness(game) { throw new Error("Must override method"); }
  isDead(game) { throw new Error("Must override method"); }
  play(game) { throw new Error("Must override method"); }
  pause(game) { throw new Error("Must override method"); }
  restart(game) { throw new Error("Must override method"); }
}
```

9.6 trexManager

Cette classe permet d'instancier et de gérer facilement le jeu du *T-Rex* de Google Chrome.

```
/**
 * Trex game manager
 * @extends gameManager
 */
class trexManager extends gameManager {
  /**
   * Create a trex manager.
   * @constructor
   */
  constructor() {
    // Parent constructor
    super();

    this.defaultTopology = [2,2,1]; //4,4,1
    this.defaultNumberOfGenomes = 12;
    this.tmpFitness = 0;
    this.lastXpos = 1000;
  }

  /**
   * Create a trex manager.
   * @return {game} The game
   */
  instanciateGame() {
    return new Runner('.interstitial-wrapper');
  }

  /**
   * Get the normalized input values
   * @param {game} game - The game
   * @return {number} The input
   */
  getNormalizedInputValues(game) {
    var inputs = [];

    if (typeof game.horizon.obstacles[0] != "undefined") {
      var maxVelocity = 13;
      var maxDistance = 600 + 25;
      var maxYPosition = 105;
      var maxSize = 3;

      var normalizedVelocity = game.currentSpeed.toFixed(3) / maxVelocity;
```

```
43        var normalizedDistance = game.horizon.obstacles[0].xPos / ↩
           maxDistance;
44        var normalizedYPosition = game.horizon.obstacles[0].yPos / ↩
           maxYPosition;
45        var normalizedSize = game.horizon.obstacles[0].size / maxSize;
46
47        inputs = [normalizedDistance,normalizedYPosition];//, ↩
           normalizedVelocity, normalizedSize
48      }
49
50      return inputs;
51    }
52
53    /**
54     * Make an action depending the result
55     * @param {game} game - The game
56     * @param {game} result - The result
57     */
58    action(game, result) {
59      // Make action depending the neural net output
60      if (result > 0.6) { // greater than 0.6 [press up]
61        simulateKeyPress(38, "keydown");
62      }
63      else if (result < 0.4) { // less than 0.4 [press down]
64        simulateKeyPress(40, "keydown");
65      }
66      else {
67        //do nothing
68      }
69    }
70
71    /**
72     * Update the fitness
73     * @param {game} game - The game
74     */
75    fitness(game) {
76      if (game.horizon.obstacles[0].xPos > this.lastXpos) {
77        this.tmpFitness++;
78      }
79      this.lastXpos = game.horizon.obstacles[0].xPos;
80    }
81
82    /**
83     * Check if the player is dead
84     * @param {game} game - The game
85     * @return {boolean} The status of the game
86     */
87    isDead(game) {
88      return game.crashed;
89    }
90
91    /**
92     * Unpause the game
93     * @param {game} game - The game
94     */
95    play(game) {
96      // Unpause the trex game
97      game.play();
98      // Simulate key press to start the game
99      simulateKeyPress(38, "keydown");
100   }
101
102   /**
103    * Pause the game
```

```
104     * @param {game} game - The game
105     */
106   pause(game) {
107     game.stop();
108   }
109
110   /**
111    * Restart the game
112    * @param {game} game - The game
113    */
114   restart(game) {
115     this.tmpFitness = 0;
116     this.lastXpos = 1000;
117     game.restart();
118   }
119 }
```

9.6.1 constructor

Le constructeur de cette classe ne prend aucun paramètre en entrée et tous les champs remplis ici doivent être faits à la main. La variable "defaultTopology" permet de renseigner la topologie par défaut du réseau de neurones du *T-Rex*. La variable "defaultNumberOfGenomes" permet de définir le nombre de génomes souhaité par génération. La variable "tmpFitness" permet de stocker temporaire le fitness du *T-Rex*. La variable "lastXpos" permet de se souvenir de la dernière position du *T-Rex* pour savoir s'il a passé un obstacle ou non.

9.6.2 instanciateGame

Permet de retourner le jeu instancié.

9.6.3 getNormalizedInputValues

Cette méthode prend en paramètre le jeu et va permettre de retourner les valeurs souhaitées normalisées. Pour commencer, on vérifie que le jeu est bel et bien instancié. Si c'est le cas on récupère toutes les valeurs que l'on souhaite, puis on les normalise en utilisant les valeurs maximum pour chaque champ. Pour finir, ces valeurs sont retournées.

9.6.4 action

Permets d'effectuer une action en fonction d'un nombre. Cette méthode prend en paramètre le jeu ainsi qu'un nombre (result) et selon le nombre, une action sera effectuée.

9.6.5 fitness

Cette méthode prend en paramètre le jeu et permet de calculer le fitness de *T-Rex*. Pour ce faire, on regarde si la position x de l'obstacle le plus proche est plus grande que la

dernière position de l'obstacle en mémoire. Si c'est le cas, alors on incrémente le fitness du *T-Rex* de 1. Après cela, on récupère la position x actuel de l'obstacle le plus proche, puis on assigne cette valeur à la variable "lastXpos".

9.6.6 isDead

Cette méthode prend en paramètre le jeu et permet de retourner le statut du jeu (mort ou vivant).

9.6.7 play

Cette méthode prend en paramètre le jeu et permet de lancer le jeu ou d'enlever une pause.

9.6.8 pause

Cette méthode prend en paramètre le jeu et permet de le mettre en pause.

9.6.9 restart

Cette méthode prend en paramètre le jeu et permet de le relancer. Elle remet également à zéro toutes les valeurs utilisées pour calculer le fitness du *T-Rex*.

9.7 flappyManager

Cette classe permet d'instancier et de gérer facilement le jeu Flappy bird.

```
1  /**
2   * Flappy game manager
3   * @extends gameManager
4   */
5  class flappyManager extends gameManager {
6    /**
7     * Create a flappy manager.
8     * @constructor
9     */
10   constructor() {
11     // Parent constructor
12     super();
13
14     this.defaultTopology = [2,2,1];
15     this.defaultNumberOfGenomes = 60;
16     this.tmpFitness = 0;
17   }
18
19   /**
20    * Create a trex manager.
21    * @return {game} The game
22    */
23   instanciateGame() {
24     return new flappyBird();
```

```
25    }
26
27    /**
28     * Get the normalized input values
29     * @param {game} game - The game
30     * @return {number} The input
31     */
32    getNormalizedInputValues(game) {
33      var inputs = [];
34
35      if (typeof game.game.backgroundSpeed != "undefined") {
36        if (game.game.pipes.length > 0) {
37          var normalizedY = game.game.birds[0].y / game.game.height;
38          var normalizedPipe;
39
40          var nextHoll = 0;
41          for(var i = 0; i < game.game.pipes.length; i+=2){
42            if(game.game.pipes[i].x + game.game.pipes[i].width > ↩
                  game.game.birds[0].x){
43              nextHoll = game.game.pipes[i].height/game.game.height;
44              break;
45            }
46          }
47
48          normalizedPipe = nextHoll;
49
50          // Create the input array for the neural network
51          inputs = [normalizedY,normalizedPipe];
52        }
53      }
54
55      return inputs;
56    }
57
58    /**
59     * Make an action depending the result
60     * @param {game} game - The game
61     * @param {game} result - The result
62     */
63    action(game, result) {
64      // Make action depending the neural net output
65      if (result > 0.5) { // greater than 0.5 [press up]
66        game.game.birds[0].flap();
67      }
68    }
69
70    /**
71     * Update the fitness
72     * @param {game} game - The game
73     */
74    fitness(game) {
75      this.tmpFitness = game.game.score;
76    }
77
78    /**
79     * Check if the player is dead
80     * @param {game} game - The game
81     * @return {boolean} The status of the game
82     */
83    isDead(game) {
84      return game.game.isItEnd();
85    }
86
87    /**
```

```
88      * Unpause the game
89      * @param {game} game - The game
90      */
91     play(game) {
92       game.game.run = true;
93     }
94
95     /**
96      * Pause the game
97      * @param {game} game - The game
98      */
99     pause(game) {
100      game.game.run = false;
101    }
102
103    /**
104     * Restart the game
105     * @param {game} game - The game
106     */
107    restart(game) {
108      this.tmpFitness = 0;
109      game.game.start();
110    }
111 }
```

9.7.1 constructor

Le constructeur de cette classe ne prend aucun paramètre en entrée et tous les champs remplis ici doivent être faits à la main. La variable "defaultTopology" permet de renseigner la topologie par défaut du réseau de neurones du *Flappy bird*. La variable "defaultNumberOfGenomes" permet de définir le nombre de génomes souhaité par génération. La variable "tmpFitness" permet de stocker temporaire le fitness du *Flappy bird*.

9.7.2 instanciateGame

Permet de retourner le jeu instancié.

9.7.3 getNormalizedInputValues

Cette méthode prend en paramètre le jeu et va permettre de retourner les valeurs souhaitées normalisées. Pour commencer, on vérifie que le jeu est bel et bien instancié. Si c'est le cas, on récupère la position Y normalisé de l'oiseau, ainsi que la taille normalisé du poteau le plus proche de l'oiseau. Pour finir, ces valeurs sont retournées.

9.7.4 action

Permets d'effectuer une action en fonction d'un nombre. Cette méthode prend en paramètre le jeu ainsi qu'un nombre (result) et selon le nombre, une action sera effectuée.

9.7.5 fitness

Cette méthode prend en paramètre le jeu et permet de calculer le fitness du *Flappy bird*. Pour ce faire, on récupère simplement le score du jeu.

9.7.6 isDead

Cette méthode prend en paramètre le jeu et permet de retourner le statut du jeu (mort ou vivant).

9.7.7 play

Cette méthode prend en paramètre le jeu et permet de lancer le jeu ou d'enlever une pause.

9.7.8 pause

Cette méthode prend en paramètre le jeu et permet de le mettre en pause.

9.7.9 restart

Cette méthode prend en paramètre le jeu et permet de le relancer. Elle remet également à zéro toutes les valeurs utilisé pour calculer le fitness du *Flappy bird*.

9.8 Generation

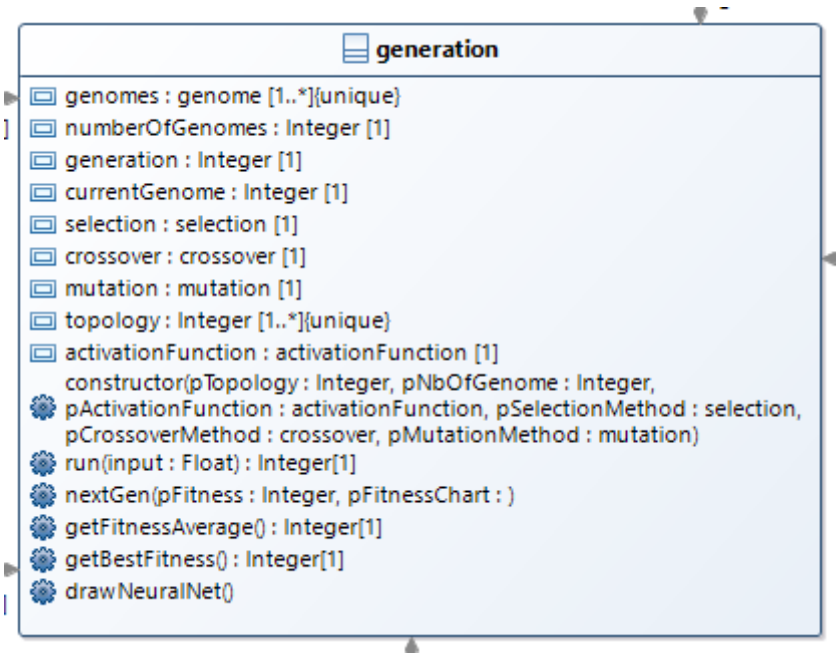


FIGURE 22 – Uml generation

Cette classe permet de créer une génération. Une génération est un objet qui va permettre de gérer plusieurs génomes.

```
1  /** Class used to create a generation. */
2  class Generation {
3    /**
4     * Create a generation.
5     * @constructor
6     * @param {number} pTopology - The topology of the neural network
7     * @param {number} pNbOfGenome - The number of genomes
8     * @param {activationFunction} pActivationFunction - The activation ↩
          function
9     * @param {selectionMethod} pSelectionMethod - The selection method
10    * @param {crossoverMethod} pCrossoverMethod - The crossover method
11    * @param {mutationMethod} pMutationMethod - The mutation method
12    */
13   constructor(pTopology, pNbOfGenome, pActivationFunction, ↩
        pSelectionMethod, pCrossoverMethod, pMutationMethod) {
14     this.genomes = [];
15     if (pNbOfGenome < 4) {
16       this.numberOfGenomes = 4;
17     }
18     else {
19       this.numberOfGenomes = pNbOfGenome;
20     }
21     this.generation = 0;
22     this.currentGenome = 0;
23
24     // Store the power for the fitness
25     this.power = 1;
26
27     // Set the different method used to create a new generation
28     this.selection = pSelectionMethod;
```

```
29      this.crossover = pCrossoverMethod;
30      this.mutation = pMutationMethod;
31
32      // Forced to store them to create a new object from the object ↩
            himself
33      this.topology = pTopology;
34      this.activationFunction = pActivationFunction;
35
36      this.bestNeuralNet = new Genome(pTopology, pActivationFunction);
37
38      // Create the new generation
39      for (var i = 0; i < this.numberOfGenomes; i++) {
40        this.genomes.push(new Genome(pTopology, pActivationFunction));
41      }
42    }
43
44    /**
45     * Run the current generation and return the output value
46     * @param {number} input - The input values
47     * @return {number} the output value.
48     */
49    run(input) {
50      // Feed the neural network with the value
51      this.genomes[this.currentGenome].feedForward(input);
52
53      // Check which move the AI should do
54      var result = this.genomes[this.currentGenome].getOutput();
55
56      return result;
57    }
58
59    /**
60     * Create the next generation
61     * @param {number} pFitness - The fitness
62     * @param {chart} pFitnessChart - The fitness chart
63     */
64    nextGen(pFitness, pFitnessChart) {
65      // Store the score of the current genome
66      this.genomes[this.currentGenome].setFitness(pFitness**this.power);
67
68      // Get the best neural net ever made
69      if (this.bestNeuralNet.fitness < pFitness**this.power) {
70        this.bestNeuralNet = this.genomes[this.currentGenome];
71      }
72
73      // Change the generation if we used all the genomes
74      if (this.currentGenome >= this.genomes.length - 1) {
75        // Reset the current genome number
76        this.currentGenome = 0;
77
78        // Change the generation
79        this.generation++;
80
81        // Add the data to the chart
82        addDataToChart(pFitnessChart,this.getBestFitness());
83
84        // Create a new generation
85        var selected = this.selection.process(this);
86        var newGen = this.crossover.process(this,selected);
87        var newMutatedGen = this.mutation.process(newGen);
88        this.genomes = newMutatedGen;
89        //this.mutation.rate =
90      }
91      else {
```

```
92        this.currentGenome++;
93      }
94    }
95
96    /**
97     * Get the fitness average of the generation
98     */
99    getFitnessAverage() {
100     var fitnessAverage = 0;
101     for (var i = 0; i < this.genomes.length; i++) {
102       fitnessAverage += this.genomes[i].fitness;
103     }
104     fitnessAverage /= this.genomes.length;
105
106     return Math.pow(fitnessAverage,1/this.power);
107   }
108
109   /**
110    * Get the best fitness of the generation
111    */
112   getBestFitness() {
113     var bestFitness = 0;
114     for (var i = 0; i < this.genomes.length; i++) {
115       if (bestFitness < this.genomes[i].fitness) {
116         bestFitness = this.genomes[i].fitness;
117       }
118     }
119
120     return Math.pow(bestFitness,1/this.power);
121   }
122
123   /**
124    * Draw the neural network
125    * @param {canvas} canvas - The canvas
126    * @param {context} context - The context of the canvas
127    */
128   drawNeuralNet(canvas,context) {
129     this.genomes[this.currentGenome].drawNeuralNet(canvas,context);
130   }
131 }
```

9.8.1 constructor

Le constructeur de cette classe prend en paramètre une topologie, un nombre de génomes, une méthode d'activation, une méthode de sélection, une méthode de crossover, ainsi qu'une méthode de mutation. Le constructeur crée un tableau contenant le nombre de génomes définit et stock tous les paramètres.

9.8.2 run

Cette méthode prend en paramètre un tableau de nombre et va s'occuper de faire passer les valeurs dans les *layers* suivants. Ensuite, la valeur du dernier *layer* est récupérée et retourner.

9.8.3 nextGen

Cette méthode prend en paramètre un fitness ainsi qu'une "chart". Elle va permettre de changer de génome ou de génération. Premièrement, le fitness temporaire qui a été calculé jusqu'à présent est assigné au fitness du génome. Après ça, on regarde si le fitness actuel est plus grand que le meilleur stocker jusqu'à présent, si c'est le cas, alors on le remplace par celui-ci. Cela permet de toujours avoir de côté le meilleur réseau de neurones.

Ensuite, on regarde si le génome actuel est le dernier de la génération. Si c'est le cas, on réinitialise la valeur stockant l'index du génome actuel, on incrémente le nombre de générations, on ajoute le meilleur fitness à la charte et on effectue toutes les méthodes de l'algorithme génétique ("selection", "crossover", "mutation"). Si ce n'est pas le cas, alors on incrémente l'index permettant de savoir sur quels génomes on se trouve.

9.8.4 getFitnessAverage

Cette méthode permet d'obtenir le fitness moyen d'une génération. Pour cela, on récupère les fitness de toute une génération, on les additionne, puis on divise le nombre obtenu par le nombre de génomes dans cette génération. Ce chiffre est ensuite retourné.

9.8.5 getBestFitness

Cette méthode permet de récupérer le meilleur fitness de la génération actuel. Pour cela, on parcourt toute la génération et à chaque fois qu'un fitness est plus gros que celui stocké, alors on le stock. Cette valeur est ensuite retournée.

9.8.6 drawNeuralNet

Cette méthode prend en paramètre un canvas et un context et permet d'exécuter la méthode "drawNeuralNet" présente sur la classe "genome".

9.9 Genome

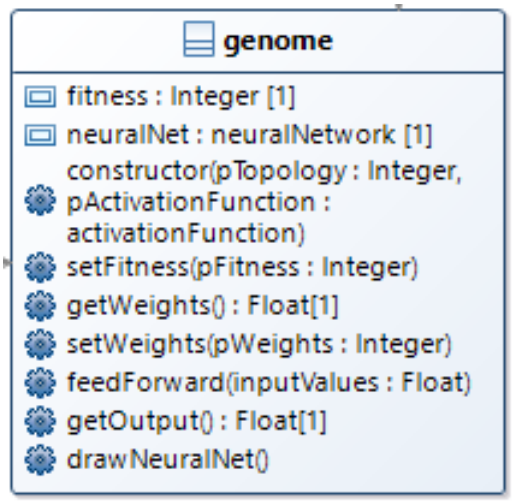


FIGURE 23 – Uml genome

Cette classe permet d'instancier un génome. Elle est principalement utilisé pour stocker le fitness d'un réseau de neurones et avoir une structure plus "logique".

```
1  /** Class used to instanciate a genome. Used to store the fitness of ↩
       a neural network */
2  class Genome {
3    /**
4     * Create a genome.
5     * @constructor
6     * @param {number} pTopology - The topology of the neural network
7     * @param {activationFunction} pActivationFunction - The activation ↩
        function
8     */
9    constructor(pTopology, pActivationFunction) {
10     this.fitness = 0;
11     this.neuralNet = new NeuralNetwork(pTopology, pActivationFunction);
12   }
13
14   /**
15    * Set the fitness of this genome
16    * @param {number} pFitness - The fitness
17    */
18   setFitness(pFitness) {
19     this.fitness = pFitness;
20   }
21
22   /**
23    * Get the fitness of this genome
24    * @return {number} the weights of the genome.
25    */
26   getWeights() {
27     return this.neuralNet.getWeights();
28   }
29
30   /**
31    * Set the weight of this genome
32    * @param {number} pWeights - The weights of the genome
33    */
```

```
34    setWeights(pWeights) {
35      this.neuralNet.setWeights(pWeights);
36    }
37
38    /**
39     * Feed forward the input values
40     * @param {number} inputValues - The input values
41     */
42    feedForward(inputValues) {
43      this.neuralNet.feedForward(inputValues);
44    }
45
46    /**
47     * Get the output of the neural network
48     * @return {number} the output value.
49     */
50    getOutput() {
51      return this.neuralNet.getOutput();
52    }
53
54    /**
55     * Draw the neural network
56     * @param {canvas} canvas - The canvas
57     * @param {context} context - The context of the canvas
58     */
59    drawNeuralNet(canvas,context) {
60      this.neuralNet.drawNeuralNet(canvas,context);
61    }
62  }
```

9.9.1 constructor

Le constructeur de cette classe prend en paramètre un tableau de nombre représentant la topologie souhaitée pour le réseau, ainsi qu'une fonction d'activation. Ce constructeur va s'occuper d'initialiser le fitness ainsi que de créer un réseau de neurones.

9.9.2 setFitness

Cette méthode va simplement servir à fixer la valeur de fitness du réseau.

9.9.3 getWeights

Cette méthode permet de récupérer tous les poids de ce réseau de neurones sous forme de tableau 1d.

9.9.4 setWeights

Cette méthode prend un tableau 1d de nombre en paramètre et permet changer tous les poids du réseau de neurones.

9.9.5 getOutput

Cette méthode permet de retourner la valeur de sortie du réseau de neurones.

9.9.6 drawNeuralNet

Cette méthode prend un canvas et un context en paramètre pour pouvoir dessiner le réseau de neurones dedans. Elle va simplement appeler la méthode "drawNeuralNet" présents sur la classe "neuralNetwork".

9.10 NeuralNetwork

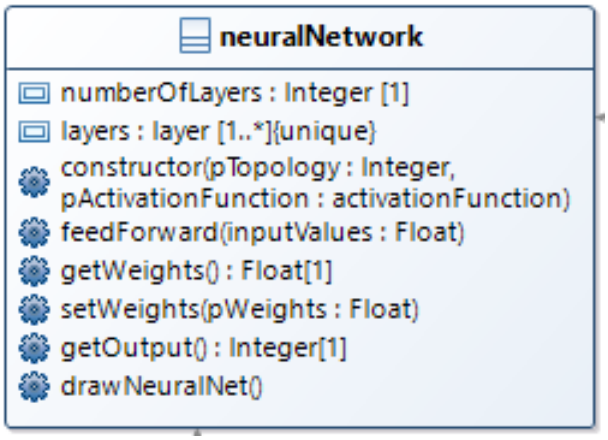


FIGURE 24 – Uml neuralNetwork

Cette classe permet d'instancier un réseau de neurones. Celui-ci va s'occuper de gérer tous les *layers* et ainsi effectuer plusieurs actions les concernant.

```
1  /** Class used to instanciate a neural network */
2  class NeuralNetwork {
3    /**
4     * Create a neural network.
5     * @constructor
6     * @param {number} pTopology - The topology of the neural network
7     * @param {activationFunction} pActivationFunction - The activation ↩
          function
8     */
9    constructor(pTopology, pActivationFunction) {
10     // Store the number of layer
11     this.numberOfLayers = pTopology.length;
12
13     // Store the layers
14     this.layers = [];
15
16     // Create the layers structure
17     for (var layerNum = 0; layerNum < this.numberOfLayers; ↩
           layerNum++) {
18       var numberOfOutputs = 0
19       // If the current layer num is not the last one
20       if (layerNum != pTopology.length - 1) {
```

```
21          // Store the size of the next layer
22          numberOfOutputs = pTopology[layerNum + 1];
23        }
24
25        // Create a new layer
26        this.layers.push(new Layer(pTopology[layerNum], ↩
              numberOfOutputs, pActivationFunction));
27      }
28    }
29
30    /**
31     * Feed forward the values
32     * @param {number} inputValues - The input values
33     */
34    feedForward(inputValues) {
35      // Assign the input values into the input neurons
36      for (var i = 0; i < inputValues.length; i++) {
37        this.layers[0].neurons[i].outputValue = inputValues[i];
38      }
39
40      // Forward propagate (start from 1 because input layer is set ↩
            before)
41      for (var layerNum = 1; layerNum < this.layers.length; layerNum++) {
42        // Store the previous layer
43        var previousLayer = this.layers[layerNum - 1];
44
45        // feed forward all the values
46        for (var neuronNum = 0; neuronNum < ↩
              this.layers[layerNum].neurons.length - ↩
              this.layers[layerNum].numberOfBias; neuronNum++) {
47          this.layers[layerNum].neurons[neuronNum].feedForward( ↩
                previousLayer);
48        }
49      }
50    }
51
52    /**
53     * Get the weights of this neural network
54     * @return {number} The weights of the neural network.
55     */
56    getWeights() {
57      var weights = [];
58      for (var i = 0; i < this.layers.length; i++) {
59        weights.push(this.layers[i].getWeights());
60      }
61      return ravel(weights);
62    }
63
64    /**
65     * Set the weights of this neural network
66     */
67    setWeights(pWeights) {
68      var pos = 0;
69      for (var i = 0; i < this.numberOfLayers; i++) {
70        var nextLayerLength = 1;
71        var nbBias = this.layers[i].numberOfBias;
72
73        // If this isn't the last layer
74        if (i+1 < this.layers.length) {//typeof this.layers[i+1] != ↩
              "undefined"
75          nbBias = this.layers[i+1].numberOfBias;
76          nextLayerLength = this.layers[i+1].neurons.length;
77        }
78
```

```
79        //Remove the number of bias from the next layer size
80        nextLayerLength -= nbBias;
81
82        //Get the number of neurons in this layer
83        var nbOfNeuronsInLayer = (this.layers[i].neurons.length) * ↩
              nextLayerLength;
84
85        //Slice the array and send the right piece at the right layer
86        this.layers[i].setWeights(pWeights.slice(pos, pos + ↩
              nbOfNeuronsInLayer), nextLayerLength);
87
88        //Increment the pos
89        pos+=nbOfNeuronsInLayer;
90      }
91    }
92
93    /**
94     * Get the result of the output neuron
95     * @return {number} The output value.
96     */
97    getOutput() {
98      return this.layers[this.layers.length - 1].neurons[0].outputValue;
99    }
100
101   /**
102    * Draw the neural network
103    */
104   drawNeuralNet(canvas,context) {
105     // The size in pixel for a neuron
106     var neuronSize = 0;
107     var neuronSpace = 10;
108     var spaceFromBorder = 20;
109     var pourcent = 15;
110
111     // Get the biggest number of neuron per layer
112     var maxNeuron = 0;
113     for (var i = 0; i < this.layers.length; i++) {
114       if (this.layers[i].neurons.length > maxNeuron) {
115         maxNeuron = this.layers[i].neurons.length;
116       }
117     }
118
119     // Get the lowest size (used to create circle)
120     if ((canvas.height / maxNeuron) < (canvas.width / ↩
          this.layers.length)) {
121       neuronSize = (canvas.height / maxNeuron);
122     }
123     else {
124       neuronSize = canvas.width / this.layers.length;
125     }
126
127     // Get the size in px for a neuron
128     neuronSize -= neuronSpace;
129
130     context.clearRect(0,0,canvas.width,canvas.height);
131
132     var spaceBetweenNeuronX = 0;
133     var spaceBetweenNeuronY = 0;
134     for (var y = 0; y < this.layers.length; y++) {
135       // Get the size between neuron [Y axis]
136       spaceBetweenNeuronY = ((canvas.width - (spaceFromBorder * 2)) / ↩
              (this.layers.length - 1)) - neuronSize / 2;
137       for (var x = 0; x < this.layers[y].neurons.length; x++) {
138         // Get the size between neuron [X axis]
```

```
139         spaceBetweenNeuronX = ((canvas.height - ↩
               (this.layers[y].neurons.length * neuronSize)) / ↩
               (this.layers[y].neurons.length + 1));
140
141         var posY = spaceBetweenNeuronY * y + spaceFromBorder + ↩
               neuronSize / 2;
142         var posX = spaceBetweenNeuronX * (x+1) + neuronSize * x + ↩
               neuronSize / 2;
143
144         // Change the color if this is the bias
145         if (x == this.layers[y].neurons.length - 1) {
146           context.fillStyle="#BDBDBD";
147           context.strokeStyle="#BDBDBD";
148         }
149         else {
150           context.fillStyle="#ee6e73";
151           context.strokeStyle="#ee6e73";
152         }
153
154         context.lineWidth=1.5;
155
156         // Draw the circle
157         context.beginPath();
158         context.arc(posY, posX, neuronSize / 2, 0, 2 * Math.PI);
159         context.stroke();
160         context.closePath();
161
162         context.font = "20px Arial";
163
164         // Draw the output values of each neuron
165         context.fillText(this.layers[y].neurons[ ↩
               x].outputValue.toFixed(3) ,posY - (context.measureText( ↩
               this.layers[y].neurons[x].outputValue.toFixed(3)).width / ↩
               2), posX + 10);
166
167         // Check if this layer is the last one
168         if (y + 1 < this.layers.length) {
169           var nextSpaceBetweenNeuronX = (canvas.height - ↩
                 (this.layers[y+1].neurons.length * neuronSize)) / ↩
                 (this.layers[y+1].neurons.length + 1);
170           for (var neuronIndex = 0; neuronIndex < ↩
                 this.layers[y+1].neurons.length - 1; neuronIndex++) {
171
172             // Get the line position
173             var linePosY = posY + neuronSize / 2;
174             var nextPosY = spaceBetweenNeuronY * (y+1) + ↩
                   spaceFromBorder;
175             var nextPosX = nextSpaceBetweenNeuronX * (neuronIndex+1) ↩
                   + neuronSize * neuronIndex + neuronSize / 2;
176
177             var yPourcent = Math.ceil(Math.ceil(linePosY - nextPosY) ↩
                   / 100 * pourcent);
178             var xPourcent = Math.ceil(Math.ceil(posX - nextPosX) / ↩
                   100 * pourcent);
179
180             context.beginPath();
181
182             // Draw the line
183             context.moveTo(linePosY, posX);
184             context.lineTo(nextPosY, nextPosX);
185             context.stroke();
186
187             // Change the font
188             context.font = "12px Arial";
```

```
189
190             // Draw a rectangle behind the text
191             context.fillStyle="#FFFFFF";
192             context.fillRect(linePosY - yPourcent - 2, posX - ↩
                    xPourcent - 11, 38, 13);
193
194             context.fillStyle="#000000";
195
196             // Draw the weight of each neuron
197             context.fillText(this.layers[y].neurons[x].outputWeights[ ↩
                    neuronIndex].weight.toFixed(3),linePosY - yPourcent, ↩
                    posX - xPourcent);
198           }
199         }
200       }
201     }
202   }
203 }
```

9.10.1 constructor

Le constructeur de cette classe prend en paramètre un tableau de nombre représentant la topologie souhaitée pour le réseau, ainsi qu'une fonction d'activation. Il va s'occuper d'instancier le bon nombre de *layer*, ainsi que stocker le nombre de *layer*.

9.10.2 feedForward

Cette méthode prend un tableau de nombre en entré et s'occupe de faire passer ces valeurs tout le long du réseau de neurones. Premièrement, les valeurs d'entrées sont assignées au *layer* "input". Ensuite, les neurones des *layers* supérieurs sont parcourus et chacun reçoit un tableau de nombre représentant le *layer* précédent. La méthode "feedForward" présente dans la classe "neuron" est alors exécutée.

9.10.3 getWeights

Cette méthode permet de récupérer les poids de tous les *layers* présents dans ce réseau de neurones. Elle retourne également ces poids en les transformant en tableau 1d si ce n'était pas déjà le cas.

9.10.4 setWeights

Cette méthode prend en paramètre un tableau 1d de nombre et s'occupe d'assigner ces nombres au différent poids du réseau de neurones. Pour ce faire, tous les *layers* sont parcourus. Pour chaque *layer*, on regarde s'il s'agit du dernier. Si c'est le cas, alors les variables contenant la taille du prochain *layer* ainsi que le nombre de bias sont respectivement fixé à 1 et au nombre de bias dans ce *layer*. Si ce n'est pas le cas, alors on récupère le nombre de bias du *layer* suivant ainsi que sa taille. On soustrait le nombre de bias à la taille du prochain *layer*. Une fois cela effectué, on calcule combien de "connection"

sont présentes entre ce *layer* et le suivant. Après ça, on découpe le tableau d'entrer pour donner les bonnes parties au bon *layer*.

9.10.5 getOutput

Cette méthode permet de retourner la valeur de sortie du réseau de neurones.

9.10.6 drawNeuralNet

Cette méthode prend en paramètre un canvas ainsi qu'un context et permet de dessiner un aperçu du réseau de neurones. Pour ce faire, on commence par obtenir le nombre de neurones le plus élever sur un *layer*. Cela permet de définir la taille maximum en pixel qu'un neurone peut avoir. Une fois ce nombre récupéré, on divise la taille en hauteur du canvas par le nombre max de neurone ainsi que la taille en largeur par le nombre de *layer* et on regarde lequel des résultats est le plus petit. Le résultat le plus petit est ensuite stocké et cela nous donne la taille en pixel que les neurones auront. Tous les *layers* sont ensuite parcourus et pour chaque *layer*, un espacement pour l'axe Y est calculé. En même temps que chaque *layer* est parcouru, on parcourt les neurones des *layers* et pour chacun d'eux, on calcule l'espacement sur l'axe X. Les positions X et Y de chaque neurone sont alors calculées. Après cela, la couleur et l'épaisseur des traits sont changés et les neurones sont dessinés. Par la suite, les trait sont dessinés. Pour cela, la position X et Y du début et de fin du trait son calculé, puis le trait est dessiné. Un pourcentage de chaque trait est calculé pour permettre de placer les poids dessus. Un rectangle est alors dessiné a l'endroit défini par le pourcentage et les poids sont dessinés par-dessus.

9.11 Layer

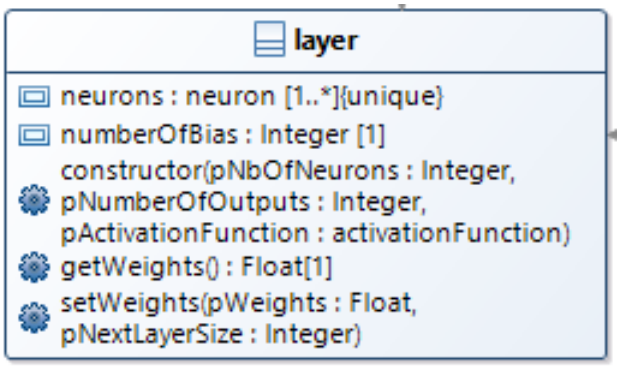


FIGURE 25 – Uml layer

Cette classe permet d'instancier un *layer* (couche du réseau de neurones). Un *layer* contient et gère des neurones, il permet d'effectuer plusieurs actions sur ceux-ci.

```
1  /** Class used to create a layer for a neural network */
2  class Layer {
3    /**
4     * Create a layer.
5     * @constructor
6     * @param {number} pNbOfNeurons - The number of neurons
7     * @param {number} pNumberOfOutputs - The number of outputs
8     * @param {activationFunction} pActivationFunction - The activation ↩
          function
9     */
10   constructor(pNbOfNeurons, pNumberOfOutputs, pActivationFunction) {
11     // Store the neurons
12     this.neurons = [];
13
14     // Store the number of bias
15     this.numberOfBias = 1;
16
17     // Create all the neurons and the bias
18     for (var neuronNum = 0; neuronNum < pNbOfNeurons + ↩
           this.numberOfBias; neuronNum++) {
19       this.neurons.push(new Neuron(pNumberOfOutputs, neuronNum, ↩
             pActivationFunction));
20     }
21   }
22
23   /**
24    * Get the weights of this layer
25    * @return {number} The weights of the layer.
26    */
27   getWeights() {
28     var weights = [];
29     for (var i = 0; i < this.neurons.length; i++) {//- this.numberOfBias
30       weights.push(this.neurons[i].getWeights());
31     }
32     return ravel(weights);
33   }
34
35   /**
36    * Set the weights of this layer
37    */
```

```
38   setWeights(pWeights, pNextLayerSize) {
39     var pos = 0;
40     for (var i = 0; i < this.neurons.length; i++) {
41       this.neurons[i].setWeights(pWeights.slice(pos, pos + ↩
             pNextLayerSize));
42       pos+=pNextLayerSize;
43     }
44   }
45 }
```

9.11.1 constructor

Le constructeur de cette classe prend en paramètre un nombre de neurones, le nombre de sortie, ainsi qu'une fonction d'activation. Il va s'occuper d'instancier tous les neurones présents dans ce *layer*.

9.11.2 getWeights

Cette méthode permet de récupérer les poids présents dans ce *layer*. Pour ce faire, tous les neurones sont parcourus un à un, puis leur poids sont récupérés et ajoutés dans un tableau. Une fois terminé, on transforme le tableau en tableau 1d si ce n'est pas déjà le cas, puis on le retourne.

9.11.3 setWeights

Cette méthode prend un tableau 1d contenant des nombres en entrée et permet de fixer les valeurs contenues dans celui-ci aux différents neurones. Pour ce faire, tous les neurones sont parcourus et chacun reçoit une partie du tableau d'entrée.

9.12 Neuron

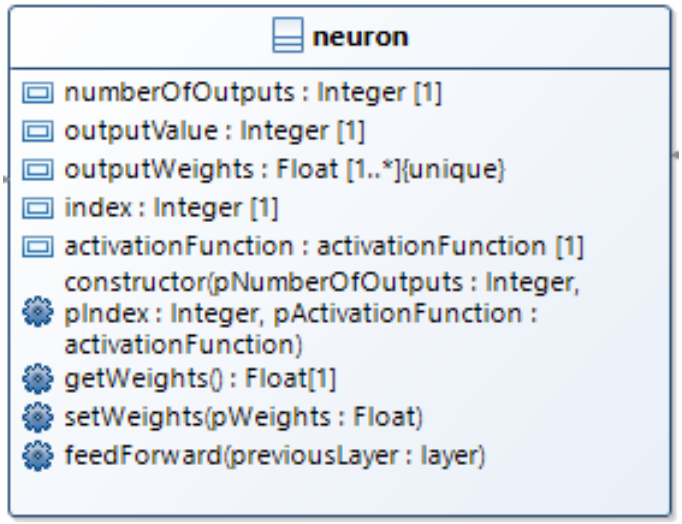


FIGURE 26 – Uml neuron

Cette classe permet d'instancier des neurones. Le neurone va s'occuper de gérer ses poids.

```
1  /** Class used to create a neuron for a neural network */
2  class Neuron {
3    /**
4     * Create a neuron.
5     * @constructor
6     * @param {number} pNumberOfOutputs - The number of outputs
7     * @param {number} pIndex - The index of this neuron
8     * @param {activationFunction} pActivationFunction - The activation ↩
          function
9     */
10   constructor(pNumberOfOutputs, pIndex, pActivationFunction) {
11     // Store the number of output
12     this.numberOfOutputs = pNumberOfOutputs;
13
14     // Store the output value
15     this.outputValue = 1;
16
17     // Store the connections
18     this.outputWeights = [];
19
20     // Create the connections
21     for (var c = 0; c < this.numberOfOutputs; c++) {
22       this.outputWeights.push(new Connection());
23     }
24
25     // Store the index of this neuron
26     this.index = pIndex;
27
28     // Store the activation function
29     this.activationFunction = pActivationFunction;
30   }
31
32   /**
33    * Get the weights of this neuron
34    * @return {number} The weigthts of this neuron.
35    */
```

```
36    getWeights() {
37      var weights = [];
38      for (var i = 0; i < this.outputWeights.length; i++) {
39        weights.push(this.outputWeights[i].getWeight());
40      }
41      return ravel(weights);
42    }
43
44    /**
45     * Set the weights of this neuron
46     * @param {number} pWeights - The weights to set
47     */
48    setWeights(pWeights) {
49      for (var c = 0; c < this.numberOfOutputs; c++) {
50        this.outputWeights[c].setWeight(pWeights[c]);
51      }
52    }
53
54    /**
55     * Feed forward the values
56     * @param {number} previousLayer - The previous layer
57     */
58    feedForward(previousLayer) {
59      var sum = 0;
60
61      //
62      for (var n = 0; n < previousLayer.neurons.length - 1; n++) {
63        sum += previousLayer.neurons[n].outputValue * ↩
              previousLayer.neurons[n].outputWeights[this.index].weight;
64      }
65
66      this.outputValue = this.activationFunction.normal(sum);
67    }
68  }
```

9.12.1 constructor

Le constructeur de cette classe prend en paramètre un nombre de sortie, l'index de ce neurone, ainsi qu'une fonction d'activation. Il va stocker toutes ces informations et créer ses connexions avec les autres neurones.

9.12.2 getWeights

Cette méthode permet de récupérer les poids présents sur le neurone. Pour ce faire, les poids de sortie sont parcourus et chaque poids est ajouté dans un tableau. Une fois cette opération terminée, on transforme le tableau en tableau 1d si ce n'est pas déjà le cas et on le retourne.

9.12.3 setWeights

Cette méthode prend en entrée un tableau de nombre et permet d'assigner ces valeurs aux différentes connexions du neurone. Pour ce faire, les connexions sont parcourues et chacune reçoit une partie du tableau d'entrer.

9.12.4 feedForward

Cette méthode prend en paramètre un tableau de nombre représentant de *layer* précèdent et permet de faire passer ces valeurs tout au long du réseau de neurones. Dans ce cas, les valeurs sont assignées au différent poids du neurone, puis la valeur de sortie du neurone est calculée avec une fonction d'activation.

9.13 Connection

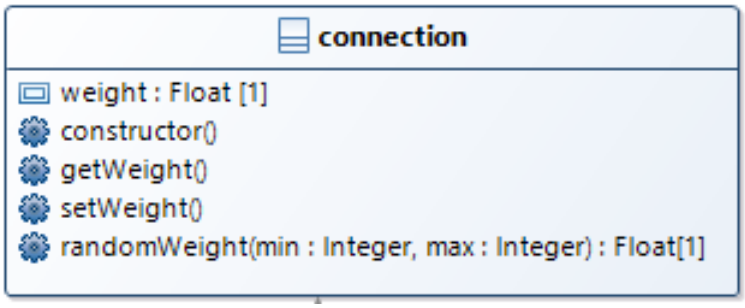


FIGURE 27 – Uml connection

Classe permet de gérer les poids pour chaque connexion de chaque neurone.

```
1  /** Class used to store the connection weight of a neuron */
2  class Connection {
3    /**
4     * Create a connection.
5     * @constructor
6     */
7    constructor() {
8      // Create the initial weight
9      this.weight = this.randomWeight(-1,1);
10
11     //this.deltaWeight;
12   }
13
14   /**
15    * Get the weight
16    * @return {number} The weights if this connection
17    */
18   getWeight() {
19     return this.weight;
20   }
21
22   /**
23    * Set the weight
24    */
25   setWeight(pWeight) {
26     this.weight = pWeight;
27   }
28
29   /**
30    * Return a random float between min an max (both include)
31    * Source : https://developer.mozilla.org/en-US/docs ↩
       /Web/JavaScript/Reference/Global_Objects/Math/random
32    * @param {number} min - The min value.
```

```
33      * @param {number} max - The max value.
34      * @return {number} The random number
35      */
36    randomWeight(min, max){
37      min = Math.ceil(min);
38      max = Math.floor(max);
39      return Math.random() * (max - min) + min;
40    }
41  }
```

9.13.1 constructor

Lance la méthode "randomWeight" pour ainsi créer un poids.

9.13.2 randomWeight

Permets de générer un nombre aléatoire entre 2 valeurs données en entrée. Les deux valeurs sont incluses.

9.14 Selection

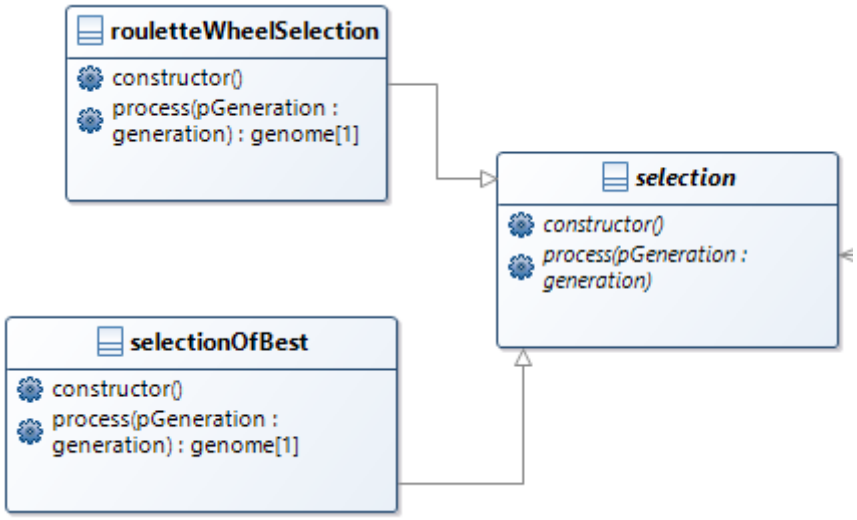


FIGURE 28 – Uml selection

Classe abstraite permettant d'avoir un modèle pour les différentes méthodes de sélection. Elle contient un constructeur qui empêche son instanciation, ainsi qu'une méthode qui permet d'effectuer la sélection. Le but est qu'en changeant la méthode de sélection lors de la création de l'objet "Generation", tout ce fasse automatiquement.

```
1  /** Abstract class for selection method */
2  class selection {
3    constructor() {
4      if (new.target === selection) {
```

```
5         throw new TypeError("Cannot construct selection instances directly");
6       }
7     }
8
9     process(pGeneration) { throw new Error("Must override method"); }
10  }
```

9.15 selectionOfBest

Cette classe permet de sélectionner les meilleurs parents. Il faut savoir que cette classe est en fait la première méthode de sélection que j'ai créée sans rien chercher sur internet et il s'avère qu'elle contient quelque bug. Elle ne devrait pas être utilisée en l'état et demandent quelques modifications.

```
1   /**
2    * NOT USED AND MUST BE CHANGED
3    * @extends selection
4    */
5   class selectionOfBest extends selection {
6     constructor() {
7       // Parent constructor
8       super();
9     }
10
11    /**
12     * Process the selection method
13     * @param {number} pGeneration - The generation.
14     * @return {number} The selected parents.
15     */
16    process(pGeneration) {
17      // Store the best genome
18      var selected = [];
19      var tmpWeight = [];
20      var genomeToAppend;
21      var max = -1;
22      var genClone;
23      var indexToRemove;
24
25      // Create a clone of the genomes
26      genClone = new Generation(pGeneration.topology, pGeneration.numberOfGenomes, pGeneration.activationFunction, pGeneration.rate);
27
28      // Get all the weights of the generation
29      for (var g = 0; g < pGeneration.genomes.length; g++) {
30        tmpWeight.push(pGeneration.genomes[g].getWeights());
31      }
32
33      // Set all the weights to the tmp generation (copy)
34      for (var g = 0; g < pGeneration.genomes.length; g++) {
35        genClone.genomes[g].setWeights(tmpWeight[g]);
36      }
37      //
38
39      // Set the fitness
40      for (var i = 0; i < pGeneration.genomes.length; i++) {
41        genClone.genomes[i].setFitness(pGeneration.genomes[i].fitness);
42      }
43
```

```
44      // Select the best genomes [2 * (sqrt(nbOfGenomes) / 2)]
45      for (var i = 0; i < ↩
            2*Math.floor(Math.sqrt(pGeneration.genomes.length)/2); i++) { ↩
            // Number of wanted genomes for breeding
46        max = -1;
47        // Check the best fitness
48        for (var j = 0; j < genClone.genomes.length; j++) {
49          // If a genome as a higher score
50          if (max < genClone.genomes[j].fitness) {
51            max = genClone.genomes[j].fitness;
52
53            // Store the best genome
54            genomeToAppend = genClone.genomes[j];
55            indexToRemove = j;
56          }
57        }
58        selected.push(genomeToAppend);
59
60        // Remove the genome from the copied array
61        genClone.genomes.splice(indexToRemove, 1);
62      }
63
64      return selected;
65    }
66  }
```

9.15.1 process

Premièrement, une copie de la génération actuelle est créée. Pour cela, une nouvelle génération vide est créée. Ensuite, tous les poids sont récupérés, puis assigner au nouveau réseau et finalement les fitness sont récupérés, puis assignés. Après avoir copié la génération, le nombre de parent souhaité est calculé (selon le nombre de génomes). Pour chaque parent sélectionné souhaité, tous les génomes sont parcourus, le génome ayant le plus gros fitness est sélectionné, stocker de coté, puis retiré de la copie de la génération. Cela permet de ne pas avoir deux fois le même réseau de neurones. Cette opération est répétée jusqu'à avoir le bon nombre de parents, puis les parents sélectionnés sont retourné.

9.16 rouletteWheelSelection

Cette classe permet de sélectionner les parents selon une méthode appelée "roulette wheel". Cette méthode pourrait être imaginée sous forme de roulette de casino. En effet, si on dit qu'un parent à 60% de la roulette, un autre 40% et que l'on joue, on a plus de chances de tomber sur le parent ayant 60%. Les parents ayant le fitness le plus élevé ont donc le plus de chance d'être sélectionnés.

```
1  /**
2   * Roulette wheel selection method
3   * @extends selection
4   */
5  class rouletteWheelSelection extends selection{
6    constructor() {
7      // Parent constructor
8      super();
9    }
```

```
10
11    /**
12     * Process the selection method
13     * @param {number} pGeneration - The generation.
14     * @return {number} The selected parents.
15     */
16    process(pGeneration) {
17      var selected = [];
18      var numberOfWantedParents = ↩
          2;//2*Math.floor(Math.sqrt(pGeneration.genomes.length)/2)
19
20      for (var i = 0; i < numberOfWantedParents; i++) { // Number of ↩
          wanted genomes for breeding
21        var sum = 0;
22        var weightSum = 0;
23
24        // Get the total fitness
25        for (var genomeIndex = 0; genomeIndex < ↩
            pGeneration.genomes.length; genomeIndex++) {
26          weightSum += pGeneration.genomes[genomeIndex].fitness + 0.1; ↩
              // +0.1 because some can have 0 fitness
27        }
28
29        // Get a random number between 0 and the weight sum
30        var threshold = Math.random() * weightSum;
31
32        // Go through each genomes
33        for (var genomeIndex = 0; genomeIndex < ↩
            pGeneration.genomes.length; genomeIndex++) {
34          sum += pGeneration.genomes[genomeIndex].fitness + 0.1;
35          // If the sum is bigger than the threshold, the current ↩
              genome is selected and removed from the array
36          if (sum > threshold) {
37            selected.push(pGeneration.genomes[genomeIndex]);
38            pGeneration.genomes.splice(genomeIndex, 1)
39            break;
40          }
41        }
42      }
43
44      return selected;
45    }
46  }
```

9.16.1 process

Premièrement, un nombre de parents souhaité est défini. Ensuite, pour chaque parent souhaité, la somme de tous les fitness de la génération est calculée, un nombre aléatoire compris entre 0 et la somme des fitness est tiré, puis les génomes sont tous parcourus un à un et leur fitness est additionné au fur et à mesure. Une fois que le cumule de fitness dépasse le nombre tiré aléatoirement, alors le génome est sélectionné en tant que parents. Cette opération est effectuée jusqu'à avoir le bon nombre de parents. Une fois cette opération effectué, les parents sélectionnés sont retournés.

9.17 Crossover

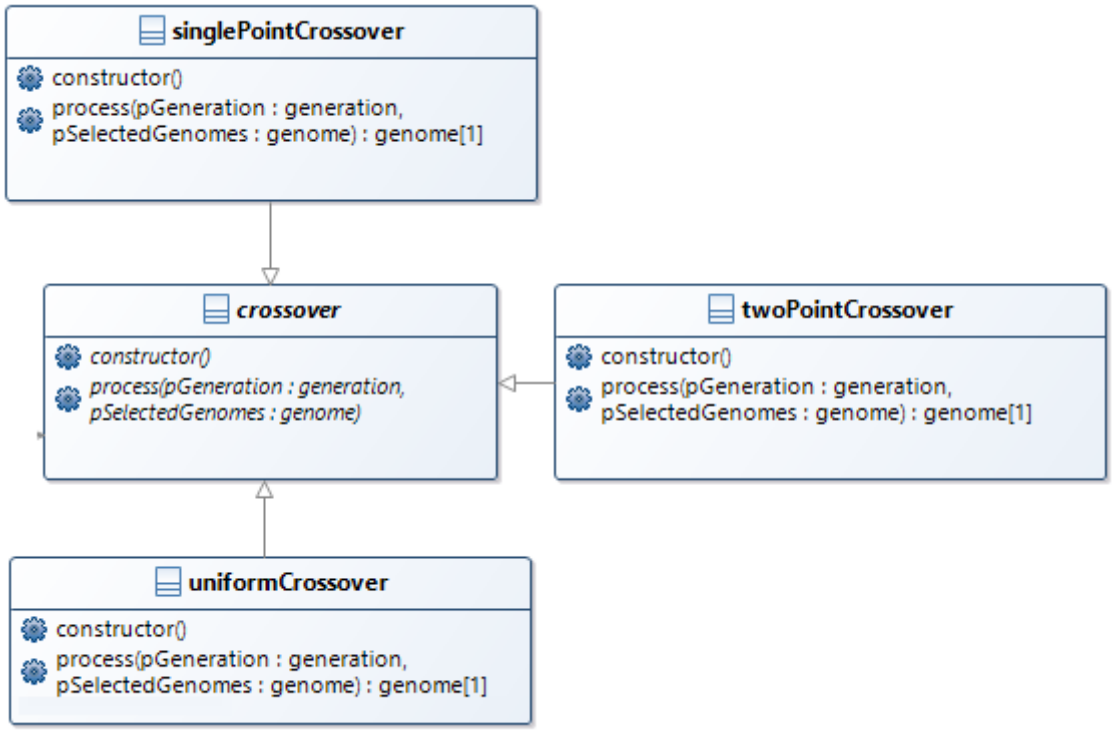


FIGURE 29 – Uml crossover

Classe abstraite permettant d'avoir un modèle pour les différentes méthodes de crossover. Elle contient un constructor qui empêche son instanciation, ainsi qu'une méthode qui permet d'effectuer le crossover. Le but est qu'en changeant la méthode de crossover lors de la création de l'objet "Generation", tout ce fasse automatiquement.

```
1  /** Abstract class for crossover method */
2  class crossover {
3    constructor() {
4      if (new.target === crossover) {
5        throw new TypeError("Cannot construct crossover instances ↩
             directly");
6      }
7    }
8
9    process(pGeneration,pSelectedGenomes) { throw new Error("Must ↩
       override method"); }
10 }
```

9.18 singlePointCrossover

Cette classe permet de faire du crossover en séparant les poids (ADN) en un point.

```
1  /**
2   * Single point crossover method
3   * @extends crossover
4   */
5  class singlePointCrossover extends crossover {
6    constructor() {
7      // Parent constructor
8      super();
9    }
10
11   /**
12    * Process the crossover method
13    * @param {number} pGeneration - The generation.
14    * @param {number} pSelectedGenomes - The selected genomes.
15    * @return {genomes} The modified genomes.
16    */
17   process(pGeneration,pSelectedGenomes) {
18     var children = [];
19     // Create a copy of the genomes
20     var genClone = new Generation(pGeneration.topology, ↩
         pGeneration.numberOfGenomes, pGeneration.activationFunction, ↩
         pGeneration.rate);
21     var weights = [];
22
23     // Store all the weights in an array
24     for (var g = 0; g < pSelectedGenomes.length; g++) {
25       weights.push(pSelectedGenomes[g].getWeights());
26     }
27
28     // For each genome minus the number of selected genomes divided ↩
         by 2
29     for (var genIndex = 0; genIndex < pGeneration.genomes.length; ↩
         genIndex++) {// - (pSelectedGenomes.length / 2)
30       //Pair of genome
31       for (var pairIndex = 0; pairIndex < pSelectedGenomes.length; ↩
           pairIndex+=2) {
32         //Get the pair of genome
33         var breedA;
34         var breedB;
35         var newWeight = [];
36         //Number of genome to create per pair
37         for (var i = 0; i < pGeneration.genomes.length / ↩
             (pSelectedGenomes.length / 2); i++) {
38           // Check which breed will be first
39           var aFirst = Math.random() >= 0.5; //Random bool
40           if (aFirst) {
41             breedA = pSelectedGenomes[pairIndex].getWeights();
42             breedB = pSelectedGenomes[pairIndex+1].getWeights();
43           }
44           else{
45             breedA = pSelectedGenomes[pairIndex+1].getWeights();
46             breedB = pSelectedGenomes[pairIndex].getWeights();
47           }
48
49           // Get a random crossover point
50           var crossoverPoint = Math.floor(Math.random() * ↩
               (weights[0].length + 1));
51
52           // Create a new weight
```

```
53            newWeight.push(breedA.slice(0, crossoverPoint));
54            newWeight.push(breedB.slice(crossoverPoint, crossoverPoint ↩
                  + weights[0].length));
55            newWeight = ravel(newWeight);
56          }
57          // Add the new weight
58          children.push(newWeight);
59        }
60
61        // Assign the new weights
62        genClone.genomes[genIndex].setWeights(children[genIndex]);
63      }
64
65      // Generate a random genomes
66      /*for (var genIndex = pGeneration.genomes.length - ↩
           (pSelectedGenomes.length / 2); genIndex < ↩
           pGeneration.genomes.length; genIndex++) {
67        genClone.genomes[genIndex] = new Genome(pGeneration.topology, ↩
             pGeneration.activationFunction);
68      }*/
69
70      // Return the new generation
71      return genClone.genomes;
72    }
73  }
```

9.18.1 process

Cette méthode permet de faire crossover en un point. Premièrement, une copie de la génération actuelle est créée. Ensuite, Toutes les générations sont parcourues. Cela va nous permettre de définir combien de nouveaux génomes doivent être créé. Pour chaque couple de parents sélectionné (il est possible de sélectionner plus de 2 parents), on regarde combien de génomes doivent être créée. Pour créer ces génomes, un booléen est tiré aléatoirement. Cela permet de définir l'ordre des parents (exemple : le parent A aura une partie de son ADN qui sera en première partie du génome créer et le parent B aura une partie de son ADN qui sera en deuxième partie.). Une fois l'ordre des parents défini, un point est tiré aléatoirement. Ce point va permettre de diviser les deux parents au bon endroit pour récupérer les bonnes parties de leur "ADN". Une fois les bonnes parties récupérées, elles sont mises ensemble, puis ajoutées dans un tableau contenant tous les nouveaux génomes. Finalement, les valeurs du tableau sont appliquées à la copie de la génération. Cette opération est répétée jusqu'à avoir le bon nombre de génomes. Une fois tout cela terminé, la nouvelle génération est retourné.

9.19 twoPointCrossover

Cette classe n'a pas été implémenté, car elle ne figurait pas dans mes priorités. En effet, les méthodes de crossover ne changent pas beaucoup le résultat final. Elle devrait permettre de, contrairement à la méthode "singlePointCrossover", séparer les poids en deux points à la place d'un.

9.20 uniformCrossover

Cette classe n'a pas été implémenté, car elle ne figurait pas dans mes priorités. En effet, les méthodes de crossover ne changent pas beaucoup le résultat final. Elle devrait permettre de, contrairement à la méthode "singlePointCrossover", séparer les poids de façon totalement aléatoire. Un à un, les poids de chaque parent ont une chance d'être sélectionnés, si le poids du parent A n'est pas sélectionné, alors le poids du parent B le sera et inversement.

9.21 Mutation

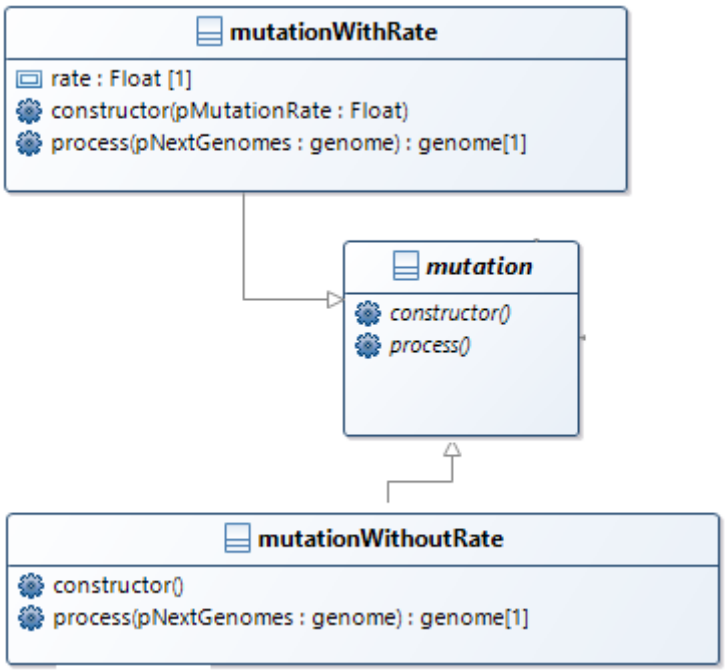


FIGURE 30 – Uml mutation

Classe abstraite permettant d'avoir un modèle pour les différentes méthodes de mutation. Elle contient un constructor qui empêche son instanciation, ainsi qu'une méthode qui permet d'effectuer la mutation. Le but est qu'en changeant la méthode de mutation lors de la création de l'objet "Generation", tout ce fasse automatiquement.

```
1  /** Abstract class for mutation method */
2  class mutation {
3    constructor() {
4      if (new.target === mutation) {
5        throw new TypeError("Cannot construct mutation instances ↩
             directly");
6      }
7    }
8
9    process(pNextGenomes) { throw new Error("Must override method"); }
10 }
```

9.22 mutationWithRate

Cette classe permet de faire de la mutation en utilisant un taux. Elle contient un constructor qui prend en paramètre un taux (entre 0 et 1) ainsi qu'une méthode qui permet de faire de la mutation avec un taux.

```
1  /**
2   * Mutation class with rate
3   * @extends mutation
4   */
5  class mutationWithRate extends mutation{
6    constructor(pMutationRate) {
7      // Parent constructor
8      super();
9
10     this.rate = pMutationRate;
11   }
12
13   /**
14    * Process the mutation method
15    * @param {number} pNextGenomes - A bunch of genomes.
16    * @return {genomes} The modified genomes.
17    */
18   process(pNextGenomes) {
19     var weights = [];
20
21     // Store all the weights in an array
22     for (var g = 0; g < pNextGenomes.length; g++) {
23       weights.push(pNextGenomes[g].getWeights());
24     }
25
26     // Apply random mutation to each genomes
27     for (var genIndex = 0; genIndex < weights.length; genIndex++) {
28
29       // Change the weights randomly
30       for (var i = 0; i < weights[genIndex].length; i++) {
31         if (this.rate <= Math.random()) {
32           weights[genIndex][i] += Math.random() * 0.4 - 0.2; // ↩
                 [-0.2...0.2]
33         }
34       }
35
36       // Apply the changes
37       pNextGenomes[genIndex].setWeights(weights[genIndex]);
38     }
39
40     return pNextGenomes;
41   }
42 }
```

9.22.1 process

Premièrement, tous les poids du prochain génome sont récupérés et stocker sous forme de tableau 1d. Ensuite, chaque poids est parcouru et a une chance d'être modifié (entre -0.2 et 0.2). Les poids sont par la suite réinsérés dans la génération, puis renvoyé.

9.23 mutationWithoutRate

Cette classe permet de faire de la mutation sans utiliser de taux. Elle contient un constructor, ainsi qu'une méthode qui permet de faire de la mutation sans taux.

```
1  /**
2   * Mutation class without rate
3   * @extends mutation
4   */
5  class mutationWithoutRate extends mutation {
6    constructor() {
7      // Parent constructor
8      super();
9    }
10
11   /**
12    * Process the mutation method
13    * @param {number} pNextGenomes - A bunch of genomes.
14    * @return {genomes} The modified genomes.
15    */
16   process(pNextGenomes) {
17     var weights = [];
18
19     // Store all the weights in an array
20     for (var g = 0; g < pNextGenomes.length; g++) {
21       weights.push(pNextGenomes[g].getWeights());
22     }
23
24     // Number of values that will change
25     var nbValuesToChange;
26
27     // Apply random mutation to each genomes
28     for (var genIndex = 0; genIndex < weights.length; genIndex++) {
29       // Get the number of values that need to change
30       var indexArray = [];
31       var alreadyExist;
32       nbValuesToChange = Math.floor(Math.random() * ↩
           (weights[genIndex].length + 1));
33
34       // While we dont have all the index that needs to change
35       while (indexArray.length < nbValuesToChange) {
36         alreadyExist = false;
37         var indexToChange = Math.floor(Math.random() * ↩
             (weights[genIndex].length + 1));
38
39         // Check if the current index has already been changed
40         for (var i = 0; i < indexArray.length; i++) {
41           if (indexToChange == indexArray[i]) {
42             alreadyExist = true;
43           }
44         }
45
46         // If the current index hasn't been changed previously
47         if (alreadyExist == false) {
48           indexArray.push(indexToChange)
49         }
50       }
51
52       // Change the weights randomly
53       for (var i = 0; i < indexArray.length; i++) {
54         weights[genIndex][indexArray[i]] += Math.random() * 0.4 - ↩
             0.2; // [-0.2...0.2]
55       }
```

```
56
57        // Apply the changes
58        pNextGenomes[genIndex].setWeights(weights[genIndex]);
59      }
60
61      return pNextGenomes;
62    }
63  }
```

9.23.1 process

Premièrement, tous les poids du prochain génome sont récupérés et stocker sous forme de tableau 1d. Ensuite, les poids récupérés sont parcourus et un nombre de poids à changer est tiré aléatoirement. Un tableau contenant les index des poids à changer est alors créer en faisant attention à ne pas avoir plusieurs fois le même index. Pour finir, les index sélectionnés sont modifiés légèrement (entre -0.2 et 0.2), les poids sont réinsérés dans le génome, puis retournés.

9.24 ActivationFunction

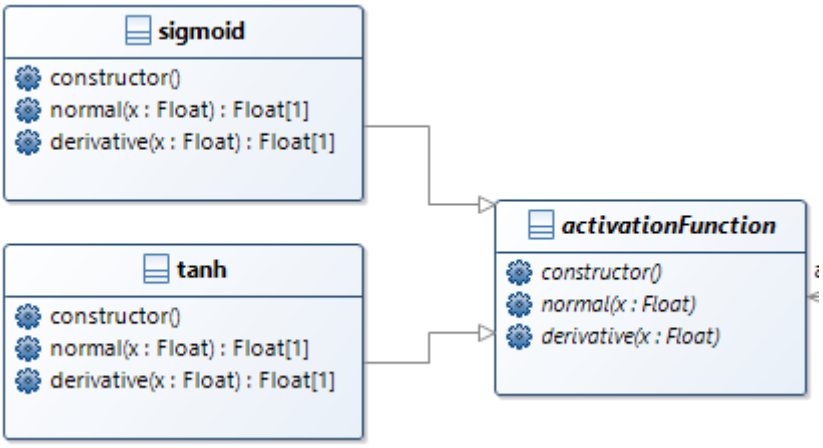


FIGURE 31 – Uml activationFunction

Classe abstraite permettant d'avoir un modèle pour les différentes fonctions d'activation. Elle contient un constructor qui empêche son instanciation ainsi que deux méthodes. La méthode "normal" qui est simplement la fonction et la méthode "derivative" qui est la fonction dérivée. Le but est qu'en changeant la fonction d'activation lors de la création de l'objet "Generation", tout ce fasse automatiquement.

```
1  /** Abstract class for activation function */
2  class activationFunction {
3    constructor() {
4      if (new.target === activationFunction) {
5        throw new TypeError("Cannot construct activationFunction ↩
             instances directly");
6      }
```

```
  }

  normal(x) { throw new Error("Must override method"); }
  derivative(x) { throw new Error("Must override method"); }
}
```

9.25 sigmoid

La classe "sigmoid" est une classe permettant d'utiliser la méthode d'activation du même nom.

```
/**
 * Sigmoid activation function class
 * @extends activationFunction
 */
class sigmoid extends activationFunction {
  /**
   * Create a sigmoid
   * @constructor
   */
  constructor() {
    // Parent constructor
    super();
  }

  /**
   * Normal function
   * @return {number} The result of the function
   */
  normal(x) {
    return 1/(1+Math.pow(Math.E, -x));
  }

  /**
   * Derivative function
   * @return {number} The result of the derivative function
   */
  derivative(x) {
    return Math.exp(-x / ((1+Math.exp(-x))**2));
  }
}
```

9.25.1 normal

Elle prend en paramètre un nombre et permet d'utiliser la fonction "normalement". Dans ce cas là, voila le calcule effectué : $\frac{1}{1+E^{-x}}$, x représentant le paramètre de la méthode et E étant une constante mathématique valant environ 2.71828.

9.25.2 derivative

Elle prend en paramètre un nombre et permet d'utiliser la fonction dérivée. Dans ce cas là, voila le calcule effectué : $E^{\frac{-x}{(1+E^{-x})^2}}$, x représentant le paramètre de la méthode et E étant une constante mathématique valant environ 2.71828.

9.26 tanh

La classe "tanh" est une classe permettant d'utiliser la méthode d'activation du même nom.

```
1  /**
2   * Tanh activation function class
3   * @extends activationFunction
4   */
5  class tanh extends activationFunction{
6    /**
7     * Create a tanh
8     * @constructor
9     */
10   constructor() {
11     // Parent constructor
12     super();
13   }
14
15   /**
16    * Normal function
17    * @return {number} The result of the function
18    */
19   normal(x) {
20     return Math.tanh(x);
21   }
22
23   /**
24    * Derivative function
25    * @return {number} The result of the derivative function
26    */
27   derivative(x) {
28     return 1 - (x * x);
29   }
30 }
```

9.26.1 normal

Elle prend en paramètre un nombre et permet d'utiliser la fonction "normalement". Dans ce cas là, il existe une méthode directement implémenté dans *Javascript* qui se nomme "tanh" et s'occupe de faire la fonction $(\tanh\left(\frac{E^x - E^{-x}}{E^x + E^{-x}}\right))$, x représentant le paramètre de la méthode et E étant une constante mathématique valant environ 2.71828.

9.26.2 derivative

Elle prend en paramètre un nombre et permet d'utiliser la fonction dérivée. Dans ce cas là, voila le calcule effectué : $1 - x^2$, x représentant le paramètre de la méthode.

10 Tests

Voici donc le résultat des tests unitaires que j'ai créé en utilisant la bibliothèque Qunit. Chaque méthode a été testé une à une pour vérifier qu'elles retournent bien ce que je souhaite. Tous les tests passent sans souci.

FIGURE 32 – Résultat des tests unitaires

11 Apport personnel

Tâche	**Apport personnel**
index.html	100%
main.js	90%, readSingleFile(), downloadFile() et removeData()
flappy.js	40%, Modification de quelque partie du code pour qu'il fonctionne dans mon projet
trex.js	10%, Modification de quelque partie du code pour qu'il fonctionne dans mon projet
function.js	100%
generation.js	100%
genome.js	100%
neuralNetwork.js	100%
layer.js	100%
neuron.js	100%
connection.js	66%, randomWeight()
activationFunction.js	100%
sigmoid.js	100%
tanh.js	100%
selection.js	100%
rouletteWheelSelection.js	100%
selectionOfBest.js	100%
crossover.js	100%
singlePointCrossover.js	100%
twoPointCrossover.js	100%
uniformCrossover.js	100%
mutation.js	100%
mutationWithRate.js	100%
mutationWithoutRate.js	100%
gameManager.js	100%
trexManager.js	100%
flappyManager.js	100%
unit_test.html	100%
unitTest.js	100%
Documentation	100%
Poster	100%

Table 1 – Apport personnel

12 Conclusion

Pour conclure, le cahier des charges de mon projet a été rempli. Hormis le fait que la version finale n'ait pas l'affichage des touches pressées, car jugé inutile et encombrant, tout est présent. Quelques fonctionnalités ont pu être ajoutées. Premièrement, le fait de pouvoir changer de jeu (choix entre le jeu du trex et flappy bird pour l'instant) a été implémenté. Le but principal de cette fonctionnalité est de montrer qu'une fois que la structure du réseau de neurones est créée, elle peut être utilisé pour beaucoup de projets. Deuxièmement, l'affichage en direct d'un aperçu du réseau de neurones a été implémenté. Cette fonction a été extrêmement utile pour déboguer et elle permet de présenter plus facilement le fonctionnement d'un réseau de neurones. Troisièmement, l'ajout d'un bouton permettant de démarrer ou mettre en pause la simulation ainsi que le jeu a été implémenté. Cette fonctionnalité a également été utile pour déboguer. Quatrièmement, l'ajout d'un graphique affichant le score de chaque génération a été implémenté. Cela permet de voir l'amélioration du réseau de neurones au fil du temps. Cinquièmement, l'exportation et l'importation de réseau de neurones ont été implémenté. Cette fonction est très utile pour par exemple sauvegarder un réseau de neurones pour pouvoir le présenter plus tard par exemple. Hormis toutes ces fonctionnalités qui ont été ajouté, le site est *responsive design*, il fonctionne donc aussi bien sur pc que sur smartphone.

Pour ce qui est des futures améliorations, plusieurs choses peuvent être envisagées. Premièrement, je compte modifier ce projet dans un futur proche pour permettre de faire du NEAT (présent dans MarI/0). Il faudrait revoir une grosse partie de la structure du code, mais c'est quelque chose qui peut être très intéressant à faire. La principale fonctionnalité d'un réseau NEAT est qu'il permet à la structure du réseau de neurones d'évoluer au fil des générations. Ensuite, implémenter plus de jeu ou bien rendre l'ajout de jeu encore plus simple peut être une piste d'amélioration. Actuellement il n'est pas compliqué d'ajouter un jeu, mais il faut modifier deux fichiers. Réduire la modification à un seul fichier serait idéal, mais je ne vois actuellement absolument pas comment faire. Pour finir, une fonctionnalité qui pourrait être rigolote plus qu'intéressante serait de permettre de modifier le réseau de neurones en live. Cette fonctionnalité n'est pas utile en soi, mais peut permettre de faire des tests.

Ce projet m'aura donc permis d'approfondir mes connaissances basiques en réseau de neurones. Il s'agit de mon premier projet utilisant le *Machine learning* ainsi que les algorithmes génétiques et grâce à cela, j'ai compris le fonctionnement en détail d'un réseau de neurones de type *supervised* et pourquoi cela fonctionne. Lors de ce projet j'ai également pu apprendre à me servir de ECMAscript6 ainsi que de Qunit. ECMAscript m'a permis de créer mes classes et Qunit à faire mes tests unitaires. Ce projet m'aura donné envie de me renseigner encore plus sur les réseaux de neurones et grâce à lui je pense apprendre à faire du NEAT ainsi que du *Reinforcement Learning*.

13 Lexique

Machine learning Ensemble de méthode permettant à une machine d'évoluer et ainsi accomplir des taches compliquées.

Algorithme génétique Permet de faire évoluer une population de solution potentiel à un problème pour donner une solution. Se base sur la sélection naturelle.

Neuroevolution Forme de *Machine learning* utilisant les algorithmes génétiques pour entraîner un réseau de neurones.

Réseau de neurones Les réseaux de neurones sont des modèles utilisés dans le *Machine learning*. Ils permettent de structurer les données.

Modèle Un modèle permet de définir la façon dont seront structurées des données.

Intelligence artificielle (*IA*) Intelligence montrée par la machine. Terme englobant l'intelligence montrée par la machine.

Supervised learning Terme utilisé lorsque l'on a des valeurs d'entrées et que l'on sait quel résultat elles doivent donner.

Unsupervised learning Terme utilisé lorsque l'on a des valeurs d'entrées, mais que l'on ne sait pas quelles valeurs de sortie on doit obtenir. Utilisé pour apprendre à traiter des données.

Reinforcement learning Terme utilisé lorsque l'on a des valeurs d'entrées, mais que l'on ne sait pas quelles valeurs de sortie on doit obtenir, on peut uniquement dire à l'algorithme si ce qu'il a fait est bien ou mal. Utilisé pour apprendre à traiter des données.

Bias Un bias est un neurones ajouté ayant une valeur fixe et qui sert a changer le fonctionnement d'une méthode d'activation.

Hyperparameter En *Machine learning*, les hyperparameter sont des paramètre que le réseau de neurones ne peut pas apprendre, il faut les renseigner a la main.

Sélection Principe des algorithmes permettant de sélectionner les individus les plus adaptés

Crossover (enjambement) Principe des algorithmes permettant de créer de nouveaux individus à partir de deux parents.

Mutation Principe des algorithmes permettant de modifier aléatoirement des parties des nouveaux individus pour éviter qu'il se ressemble tous.

Feed forward Principe du *Machine learning* consistant à faire circuler les informations d'entrée jusqu'à la sortie.

Fitness Terme utilisé pour designer un "score". Dans le cas où on cherche à avoir le score le plus élevé, le terme "fitness" sera utilisé.

Cost Terme utilisé pour désigner un "score". Dans le cas où on cherche à avoir le score le plus petit possible, le terme "cost" sera utilisé.

Poids (weights) Les poids permettent de définir la force de la connection entre deux neurones

Fonction d'activation Fonction mathématique permettant de transformer un nombre en signal de sortie.

NEAT Évolution de la *Neuroevolution* (*Neuroevolution of augmenting topologies*). Permet de modifier les poids et la structure du réseau de neurones.

Glitch Anomalie dans le moteur de gestion de la physique dans un jeu vidéo.

Cloud Computing Exploitation de la puissance de calcul ou de stockage de serveurs informatiques distants.

14 Sources

Références

[1] Wikipedia. *Intelligence artificielle.*
URL : https://fr.wikipedia.org/wiki/Intelligence_artificielle.

[2] Xviniette. *Flappy learning.*
URL : https://github.com/xviniette/FlappyLearning.

[3] Oriol Vinyals. *Starcraft IA.*
URL : https://deepmind.com/blog/deepmind-and-blizzard-release-starcraft-ii-ai-research-environment/.

[4] SethBling. *MarI/O.*
URL : https://www.youtube.com/watch?v=qv6UVOQ0F44.

[5] Mobilegeeks. *Forza Drivatar.*
URL : https://www.youtube.com/watch?v=twI0RSVwnR0.

[6] Xbox Wire Staff. *Forza Drivatar.*
URL : https://news.xbox.com/2014/09/30/games-forza-horizon-2-drivatars/.

[7] Kyle Orland. *Forza Drivatar.*
URL : https://arstechnica.com/gaming/2013/10/how-forza-5-and-the-xbox-one-use-the-cloud-to-drive-machine-learning-ai/.

[8] TASVideos. *BizHawk.*
URL : https://github.com/TASVideos/BizHawk.

[9] Wikipedia. *NEAT.*
URL : https://en.wikipedia.org/wiki/Neuroevolution_of_augmenting_topologies.

[10] Wikipedia. *Définition glitch.*
URL : https://fr.wikipedia.org/wiki/Glitch.

[11] Wikipedia. *Cloud computing.*
URL : https://fr.wikipedia.org/wiki/Cloud_computing.

[12] Wikipedia. *Alpha Go.*
URL : https://fr.wikipedia.org/wiki/AlphaGo.

[13] Wikipedia. *Reinforcement Learning.*
URL : https://fr.wikipedia.org/wiki/Apprentissage_par_renforcement.

[14] Burak KANBER. *Machine Learning : Introduction to Genetic Algorithms.*
URL : https://www.burakkanber.com/blog/machine-learning-genetic-algorithms-part-1-javascript/.

[15] DOMENIC. *Classe abstraite en Javascript.*
URL : http://stackoverflow.com/questions/29480569/does-ecmascript-6-have-a-convention-for-abstract-classes.

[16] WIKIPEDIA. *Roulette wheel selection.*
URL : https://en.wikipedia.org/wiki/Fitness_proportionate_selection.

[17] Qasim Mohammed HUSSEIN. *Elitism.*
URL : https://www.researchgate.net/post/What_is_meant_by_the_term_Elitism_in_the_Genetic_Algorithm.

[18] QUNIT. *Qunit.*
URL : https://qunitjs.com/.

Table des figures

Liste des tableaux

15 Code source

15.1 unit_test HTML

```
<!DOCTYPE html>
<html>

  <head>
   <meta charset="utf-8">
   <title>QUnit Example</title>

   <!-- QUNIT CSS -->
   <link rel="stylesheet" href="./libraries/qunit/qunit-2.3.2.css">
  </head>

  <body>
   <div id="qunit"></div>
   <div id="qunit-fixture"></div>

   <!-- QUNIT JS -->
   <script src="./libraries/qunit/qunit-2.3.2.js"></script>

   <!-- APPLICATION SCRIPTS -->
   <script src="./classes/neural_network.js"></script>
   <script src="./classes/layer.js"></script>
   <script src="./classes/neuron.js"></script>
   <script src="./classes/connection.js"></script>
   <script src="./classes/activation_function.js"></script>
   <script src="./classes/generation.js"></script>
   <script src="./classes/genome.js"></script>
   <script src="./classes/selection.js"></script>
   <script src="./classes/crossover.js"></script>
   <script src="./classes/mutation.js"></script>
   <script src="./function.js"></script>

   <!-- UNIT TEST FILE -->
   <script src="unitTest.js"></script>
  </body>

</html>
```

15.2 unitTest JS

```
//
QUnit.test('Generation constructor', function (assert) {
  assert.expect(6);

  var topology = [2,2,1];
  var numberOfGenomes = 12;
  var activation = new sigmoid();
  var selectionMethod = new rouletteWheelSelection();
  var crossoverMethod = new singlePointCrossover();
  var mutationMethod = new mutationWithRate(0.2);

  var generation = new Generation(topology, numberOfGenomes, ↩
   activation, selectionMethod, crossoverMethod, mutationMethod);

  assert.deepEqual(generation.topology, topology, undefined);
  assert.equal(generation.numberOfGenomes, numberOfGenomes, undefined);
```

```
16    assert.equal(generation.activationFunction, activation, undefined);
17    assert.equal(generation.selection, selectionMethod, undefined);
18    assert.equal(generation.crossover, crossoverMethod, undefined);
19    assert.equal(generation.mutation, mutationMethod, undefined);
20  });
21
22  //
23  QUnit.test('Generation run', function (assert) {
24    assert.expect(1);
25
26    var topology = [2,2,1];
27    var numberOfGenomes = 12;
28    var activation = new sigmoid();
29    var selectionMethod = new rouletteWheelSelection();
30    var crossoverMethod = new singlePointCrossover();
31    var mutationMethod = new mutationWithRate(0.2);
32
33    var input = [1,1];
34
35    var generation = new Generation(topology, numberOfGenomes, ↩
       activation, selectionMethod, crossoverMethod, mutationMethod);
36
37    assert.equal(typeof generation.run(input), "number", undefined);
38  });
39
40  //
41  QUnit.test('Generation getFitnessAverage', function (assert) {
42    assert.expect(1);
43
44    var topology = [2,2,1];
45    var numberOfGenomes = 12;
46    var activation = new sigmoid();
47    var selectionMethod = new rouletteWheelSelection();
48    var crossoverMethod = new singlePointCrossover();
49    var mutationMethod = new mutationWithRate(0.2);
50
51    var generation = new Generation(topology, numberOfGenomes, ↩
       activation, selectionMethod, crossoverMethod, mutationMethod);
52
53    assert.equal(typeof generation.getFitnessAverage(), "number", ↩
       undefined);
54  });
55
56  //
57  QUnit.test('Generation getBestFitness', function (assert) {
58    assert.expect(1);
59
60    var topology = [2,2,1];
61    var numberOfGenomes = 12;
62    var activation = new sigmoid();
63    var selectionMethod = new rouletteWheelSelection();
64    var crossoverMethod = new singlePointCrossover();
65    var mutationMethod = new mutationWithRate(0.2);
66
67    var generation = new Generation(topology, numberOfGenomes, ↩
       activation, selectionMethod, crossoverMethod, mutationMethod);
68
69    assert.equal(typeof generation.getBestFitness(), "number", undefined);
70  });
71
72  //
73  QUnit.test('Genome constructor', function (assert) {
74    assert.expect(1);
75
```

```
76    var topology = [2,2,1];
77    var activation = new sigmoid();
78
79    var genome = new Genome(topology, activation);
80
81    assert.notEqual(genome.neuralNet, null, undefined);
82  });
83
84  //
85  QUnit.test('Genome setFitness', function (assert) {
86    assert.expect(1);
87
88    var topology = [2,2,1];
89    var activation = new sigmoid();
90
91    var genome = new Genome(topology, activation);
92
93    genome.setFitness(1)
94    assert.equal(genome.fitness, 1, undefined);
95  });
96
97  //
98  QUnit.test('Genome getWeights', function (assert) {
99    assert.expect(1);
100
101   var topology = [2,2,1];
102   var activation = new sigmoid();
103
104   var genome = new Genome(topology, activation);
105
106   assert.equal(typeof genome.getWeights(), "object", undefined);
107 });
108
109 //
110 QUnit.test('Genome setWeights', function (assert) {
111   assert.expect(1);
112
113   var topology = [1,1,1];
114   var activation = new sigmoid();
115
116   var genome = new Genome(topology, activation);
117   var weights = [1,2,3,4];
118   genome.setWeights(weights);
119
120   assert.deepEqual(genome.getWeights(), weights, undefined);
121 });
122
123 //
124 QUnit.test('Genome getOutput', function (assert) {
125   assert.expect(1);
126
127   var topology = [1,1,1];
128   var activation = new sigmoid();
129
130   var genome = new Genome(topology, activation);
131
132   assert.equal(typeof genome.getOutput(), "number", undefined);
133 });
134
135 //
136 QUnit.test('NeuralNetwork constructor', function (assert) {
137   assert.expect(2);
138
139   var topology = [1,1,1];
```

```
140   var activation = new sigmoid();
141
142   var neuralNetwork = new NeuralNetwork(topology, activation);
143
144   assert.equal(neuralNetwork.numberOfLayers, topology.length, ↩
       undefined);
145   assert.equal(neuralNetwork.layers.length, topology.length, undefined);
146 });
147
148 //
149 QUnit.test('NeuralNetwork getWeights', function (assert) {
150   assert.expect(1);
151
152   var topology = [1,1,1];
153   var activation = new sigmoid();
154
155   var neuralNetwork = new NeuralNetwork(topology, activation);
156
157   assert.equal(typeof neuralNetwork.getWeights(), "object", undefined);
158 });
159
160 //
161 QUnit.test('NeuralNetwork setWeights', function (assert) {
162   assert.expect(1);
163
164   var topology = [1,1,1];
165   var activation = new sigmoid();
166
167   var neuralNetwork = new NeuralNetwork(topology, activation);
168
169   var weights = [1,2,3,4];
170   neuralNetwork.setWeights(weights);
171
172   assert.deepEqual(neuralNetwork.getWeights(), weights, undefined);
173 });
174
175 //
176 QUnit.test('NeuralNetwork getOutput', function (assert) {
177   assert.expect(1);
178
179   var topology = [1,1,1];
180   var activation = new sigmoid();
181
182   var neuralNetwork = new NeuralNetwork(topology, activation);
183
184   assert.equal(typeof neuralNetwork.getOutput(), "number", undefined);
185 });
186
187 //
188 QUnit.test('Layer constructor', function (assert) {
189   assert.expect(1);
190
191   var numberOfNeurons = 1;
192   var numberOfOutputs = 2;
193   var activation = new sigmoid();
194
195   var layer = new Layer(numberOfNeurons, numberOfOutputs, activation);
196
197   assert.equal(layer.neurons.length, numberOfOutputs, undefined);
198 });
199
200 //
201 QUnit.test('Layer getWeights', function (assert) {
202   assert.expect(1);
```

```
203
204   var numberOfNeurons = 1;
205   var numberOfOutputs = 2;
206   var activation = new sigmoid();
207
208   var layer = new Layer(numberOfNeurons, numberOfOutputs, activation);
209
210   var weights = [1,2,3,4];
211   layer.setWeights(weights, 2);
212
213   assert.deepEqual(layer.getWeights(), weights, undefined);
214 });
215
216 //
217 QUnit.test('Layer setWeights', function (assert) {
218   assert.expect(1);
219
220   var numberOfNeurons = 1;
221   var numberOfOutputs = 2;
222   var activation = new sigmoid();
223
224   var layer = new Layer(numberOfNeurons, numberOfOutputs, activation);
225
226   var weights = [1,2,3,4];
227   layer.setWeights(weights, 2);
228
229   assert.deepEqual(layer.getWeights(), weights, undefined);
230 });
231
232 //
233 QUnit.test('Neuron constructor', function (assert) {
234   assert.expect(4);
235
236   var numberOfOutputs = 2;
237   var index = 1;
238   var activation = new sigmoid();
239
240   var neuron = new Neuron(numberOfOutputs, index, activation);
241
242   assert.equal(neuron.numberOfOutputs, numberOfOutputs, undefined);
243   assert.equal(neuron.index, index, undefined);
244   assert.equal(neuron.activationFunction, activation, undefined);
245   assert.equal(neuron.outputWeights.length, numberOfOutputs, undefined);
246 });
247
248 //
249 QUnit.test('Neuron getWeights', function (assert) {
250   assert.expect(1);
251
252   var numberOfOutputs = 2;
253   var index = 1;
254   var activation = new sigmoid();
255
256   var neuron = new Neuron(numberOfOutputs, index, activation);
257
258   var weights = [1,2];
259   neuron.setWeights(weights);
260
261   assert.deepEqual(neuron.getWeights(), weights, undefined);
262 });
263
264 //
265 QUnit.test('Neuron setWeights', function (assert) {
266   assert.expect(1);
```

```
267
268   var numberOfOutputs = 2;
269   var index = 1;
270   var activation = new sigmoid();
271
272   var neuron = new Neuron(numberOfOutputs, index, activation);
273
274   var weights = [1,2];
275   neuron.setWeights(weights);
276
277   assert.deepEqual(neuron.getWeights(), weights, undefined);
278 });
279
280 //
281 QUnit.test('Connection constructor', function (assert) {
282   assert.expect(1);
283
284   var connection = new Connection();
285
286   assert.equal(typeof connection.weight, "number", undefined);
287 });
288
289 //
290 QUnit.test('Connection getWeight', function (assert) {
291   assert.expect(1);
292
293   var number = 2;
294   var connection = new Connection();
295   connection.setWeight(number);
296
297   assert.equal(connection.getWeight(), number, undefined);
298 });
299
300 //
301 QUnit.test('Connection setWeight', function (assert) {
302   assert.expect(1);
303
304   var number = 2;
305   var connection = new Connection();
306   connection.setWeight(number);
307
308   assert.equal(connection.getWeight(), number, undefined);
309 });
310
311 //
312 QUnit.test('Connection randomWeight', function (assert) {
313   assert.expect(1);
314
315   var connection = new Connection();
316
317   assert.equal(typeof connection.getWeight(), "number", undefined);
318 });
319
320 //
321 QUnit.test('rouletteWheelSelection process', function (assert) {
322   assert.equal(true,true, undefined);
323 });
324
325 //
326 QUnit.test('singlePointCrossover process', function (assert) {
327   assert.equal(true,true, undefined);
328 });
329
330 //
```

```
QUnit.test('mutationWithoutRate process', function (assert) {
  assert.equal(true,true, undefined);
});

//
QUnit.test('mutationWithRate process', function (assert) {
  assert.equal(true,true, undefined);
});

//
QUnit.test('sigmoid normal', function (assert) {
  assert.expect(1);

  var sig = new sigmoid();

  assert.equal(sig.normal(1), 0.7310585786300049, undefined);
});

//
QUnit.test('sigmoid derivative', function (assert) {
  assert.expect(1);

  var sig = new sigmoid();

  assert.equal(sig.derivative(1), 0.5859934626301939, undefined);
});

//
QUnit.test('tanh normal', function (assert) {
  assert.expect(1);

  var tan = new tanh();

  assert.equal(tan.normal(1), 0.7615941559557649, undefined);
});

//
QUnit.test('tanh derivative', function (assert) {
  assert.expect(1);

  var tan = new tanh();

  assert.equal(tan.derivative(1), 0, undefined);
});
```

15.3 index

```
<!doctype html>
<html>

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, ↩
   initial-scale=1.0,maximum-scale=1.0, user-scalable=no">
  <title>T-Rex Ceptional</title>
  <link rel="icon" type="image/png" sizes="32x32" ↩
   href="assets/favicon.png">

  <link rel="stylesheet" href="css.css">
  <link type="text/css" rel="stylesheet" ↩
   href="libraries/materialize/css/materialize.min.css" ↩
```

```
    media="screen,projection"/>
12  <link ↩
    href="https://fonts.googleapis.com/icon?family=Material+Icons" ↩
    rel="stylesheet">
13  </head>
14
15  <body id="t" class="offline">
16      <nav>
17        <div class="nav-wrapper">
18          <a href="#" class="brand-logo" style="margin-left: ↩
    10px;">T-Rex Ceptional</a>
19          <ul id="nav-mobile" class="right hide-on-med-and-down">
20            <li><a href="#aboutModal">About</a></li>
21            <!--<li><a href="#contactModal">Contact</a></li>-->
22          </ul>
23        </div>
24      </nav>
25
26    <div class="container">
27
28      <div class="row">
29
30        <div class="col s12 m12 l2">
31          <div class="card-panel">
32            <h5>Informations</h5>
33            <div class="card-content">
34              <div class="card-panel grey lighten-2">
35                <h5>Network infos</h5>
36                <div class="card-content">
37                  <ul>
38                    <li>Generation : <span ↩
    id="generationIndex">X</span></li>
39                    <li>Genome : <span id="genomeIndex">X</span></li>
40                    <li>Time passed [s] : <span ↩
    id="timeIndex">0</span></li>
41                  </ul>
42                </div>
43              </div>
44              <div class="card-panel grey lighten-2">
45                <h5>Game selector</h5>
46                <select name="gameSelector">
47                  <option value="1" selected>T-Rex</option>
48                  <option value="2">Flappy bird</option>
49                </select>
50              </div>
51              <!--<div class="card-panel grey lighten-2" id="gameValues">
52                <h5>T-Rex values</h5>
53                <div class="card-content">
54                  <ul>
55                    <li>Distance : <span id="Distance">X</span></li>
56                    <li>Y position : <span id="yPosition">X</span></li>
57                    <li>Velocity : <span id="Velocity">X</span></li>
58                    <li>Size : <span id="Size">X</span></li>
59                  </ul>
60                </div>
61              </div>-->
62              <button class="btn waves-effect waves-light" ↩
    type="button" id="stateBtn" onclick="changeRunningState();">START
63                <i class="material-icons right">power_settings_new</i>
64              </button>
65            </div>
66          </div>
67        </div>
68
```

```
69        <div class="col s12 m12 l6">
70          <div class="col s12 m12 l12">
71            <div class="card-panel">
72              <h5>Game</h5>
73              <div class="card-content">
74                <div id="main-frame-error" class="interstitial-wrapper">
75                  <div id="main-content">
76                    <div class="icon icon-offline" alt=""></div>
77                  </div>
78                  <div id="offline-resources">
79                    <img id="offline-resources-1x" ↩
    src="assets/default_100_percent/100-offline-sprite.png">
80                    <img id="offline-resources-2x" ↩
    src="assets/default_200_percent/200-offline-sprite.png">
81                  </div>
82                </div>
83              </div>
84            </div>
85          </div>
86
87          <div class="col s12 m12 l12">
88            <div class="card-panel" id="neuralNetPreview">
89              <canvas id="neuralNet" width="710" height="400" ↩
    style="border:1px solid #d3d3d3;"></canvas>
90            </div>
91          </div>
92        </div>
93
94        <div class="col s12 m12 l4">
95          <div class="col s12 m12 l12">
96            <div class="card-panel">
97              <canvas id="Chart" width="400" height="400"></canvas>
98            </div>
99          </div>
100       </div>
101
102     </div>
103
104     </div>
105
106     <!-- Toolbar -->
107     <div class="fixed-action-btn toolbar">
108       <a class="btn-floating btn-large pulse red lighten-2">
109         <i class="large material-icons">dashboard</i>
110       </a>
111       <ul>
112         <li class="waves-effect waves-light">
113           <a class="modal-trigger waves-effect waves-light btn" ↩
    href="#exportModal" onclick="resfreshNNselection()">EXPORT</a>
114         </li>
115         <li class="waves-effect waves-light">
116           <a class="modal-trigger waves-effect waves-light btn" ↩
    href="#importModal">IMPORT</a>
117         </li>
118         <!--<li class="waves-effect waves-light"><a ↩
    class="modal-trigger waves-effect waves-light btn" ↩
    href="#showNetModal">SHOW/MODIFY NEURAL NETWORK</a></li>-->
119       </ul>
120     </div>
121
122     <!-- Export modal -->
123     <div id="exportModal" class="modal modal-footer">
124       <div class="modal-content">
125         <h4>Export the neural network</h4>
```

```
126
127         <div class="row">
128           <div class="col s6 m6 l6">
129             <label for="neuralNetSelector">Select the neural net you want to export</label>
130             <select id="neuralNetSelector" style="max-height: 65px;">
131               <option value="1" selected>Neural net 1</option>
132               <option value="2">Neural net 2</option>
133             </select>
134           </div>
135           <div class="col s6 m6 l6">
136             <button class="btn waves-effect waves-light" type="submit" name="action" onclick="getBestNN()">Export the best neural net
137               <i class="material-icons right">send</i>
138             </button>
139           </div>
140         </div>
141
142         <hr/>
143
144         <div class="row">
145           <div class="col s12 m12 l12">
146             <form onsubmit="downloadFile('neural_network.txt', this['text'].value); return false;">
147               <div class="input-field col s12">
148                 <label for="textareaExport">Exported to text</label>
149                 <textarea id="textareaExport" name="text" class="materialize-textarea" placeholder="Exported to text"></textarea>
150               </div>
151
152               <button class="btn waves-effect waves-light" type="submit" name="action">EXPORT TO FILE
153                 <i class="material-icons right">send</i>
154               </button>
155             </form>
156           </div>
157         </div>
158       </div>
159       <div class="modal-footer">
160         <a href="#!" class="modal-action modal-close waves-effect waves-red btn-flat ">Close</a>
161       </div>
162     </div>
163
164     <!-- Import modal -->
165     <div id="importModal" class="modal modal-footer">
166       <div class="modal-content">
167         <h4>Import a neural network</h4>
168
169         <div class="file-field input-field">
170           <div class="btn">
171             <span>SELECT FILE</span>
172             <input type="file" id="selectedFile">
173           </div>
174           <div class="file-path-wrapper">
175             <input class="file-path validate" type="text">
176           </div>
177         </div>
178
179         <div class="input-field col s12">
180           <textarea id="textareaImport" name="text" class="materialize-textarea" placeholder="Text to import"></textarea>
```

```
181         </div>
182
183         <button class="btn waves-effect waves-light" type="button" name="action" onclick="importNeuralNetwork();">IMPORT NEURAL NETWORK
184           <i class="material-icons right">send</i>
185         </button>
186       </div>
187       <div class="modal-footer">
188         <a href="#!" class="modal-action modal-close waves-effect waves-red btn-flat ">Close</a>
189       </div>
190     </div>
191
192     <!-- Contact modal -->
193     <!--<div id="contactModal" class="modal modal-footer">
194       <div class="modal-content">
195         <h4>Contact</h4>
196         <p>A bunch of text</p>
197       </div>
198       <div class="modal-footer">
199         <a href="#!" class="modal-action modal-close waves-effect waves-red btn-flat ">Close</a>
200       </div>
201     </div>-->
202
203     <!-- About modal -->
204     <div id="aboutModal" class="modal modal-footer">
205       <div class="modal-content">
206         <h4>About</h4>
207         <p>
208         The purpose of this project is to create an <i>Artificial intelligence</i> that aims to learn to play several video games. It implements some principles of machine learning such as neural networks, as well as genetic algorithms.</br></br>
209
210         A neural network of the <i>supervised</i> type will allow the creation of the structure of the brain used by the <i>Artificial intelligence</i>, while the genetic algorithm will train it. Each brain will run in a simulation of a game and will get a score when the game ends. This score will be used to select the best brains. Those brains will then reproduce and "crossover", thus merging and mixing values.</br></br>
211
212         The neural network takes normalized values and inputs from the game as well as "weights" that pass into mathematical functions. Those functions determine what action to make. Once each neural network has been used in a simulation, we take the ones with the biggest score to interchange their "weight" and modify them slightly. Thoses operations are repeated infinitely.</br></br>
213
214         This project allows to add new games by refering some values and gets a very good score on the games already implemented.
215         </p>
216       </div>
217       <div class="modal-footer">
218         <a href="#!" class="modal-action modal-close waves-effect waves-red btn-flat ">Close</a>
219       </div>
220     </div>
221
222     <div style="display:none;">
223       <img id="flapBackground" src="./assets/flappy/background.png">
224       <img id="flapBird" src="./assets/flappy/bird.png">
```

```
225         <img id="flapPipeBottom" src="./assets/flappy/pipebottom.png">
226         <img id="flapPipeTop" src="./assets/flappy/pipetop.png">
227       </div>
228
229     <!-- SCRIPTS -->
230     <script src="libraries/jQuery/jquery-3.2.1.min.js"></script>
231     <script src="libraries/chartJS/Chart.js"></script>
232     <script type="text/javascript" ↩
        src="libraries/materialize/js/materialize.min.js"></script>
233
234     <script src="classes/neural_network.js"></script>
235     <script src="classes/layer.js"></script>
236     <script src="classes/neuron.js"></script>
237     <script src="classes/connection.js"></script>
238     <script src="classes/activation_function.js"></script>
239     <script src="classes/generation.js"></script>
240     <script src="classes/genome.js"></script>
241     <script src="classes/selection.js"></script>
242     <script src="classes/crossover.js"></script>
243     <script src="classes/mutation.js"></script>
244     <script src="classes/gameManager.js"></script>
245
246     <script src="function.js"></script>
247     <script src="trex.js"></script>
248     <script src="flappy.js"></script>
249     <script src="main.js"></script>
250 </body>
251
252 </html>
```

15.4 css

```
1  /* Copyright 2013 The Chromium Authors. All rights reserved.
2   * Use of this source code is governed by a BSD-style license that ↩
       can be
3   * found in the LICENSE file. */
4
5  html, body {
6   padding: 0;
7   margin: 0;
8   width: 100%;
9   height: 100%;
10  background-color: #4db6ac;
11 }
12
13 .icon {
14  -webkit-user-select: none;
15  user-select: none;
16  display: inline-block;
17 }
18
19 .icon-offline {
20  content: -webkit-image-set( ↩
      url(assets/default_100_percent/100-error-offline.png) 1x, ↩
      url(assets/default_200_percent/200-error-offline.png) 2x);
21  position: relative;
22 }
23
24 .hidden {
25  display: none;
26 }
```

```
27
28
29 /* Offline page */
30
31 .offline .interstitial-wrapper {
32  color: #2b2b2b;
33  font-size: 1em;
34  line-height: 1.55;
35  margin: 0 auto;
36  max-width: 600px;
37  /*padding-top: 100px;*/
38  width: 100%;
39 }
40
41 .offline .runner-container {
42  height: 150px;
43  max-width: 600px;
44  overflow: hidden;
45  /*position: absolute;*/
46  /*top: 35px;*/
47  width: 44px;
48  /*margin-top: 150px;*/
49 }
50
51 .offline .runner-canvas {
52  height: 150px;
53  max-width: 600px;
54  opacity: 1;
55  overflow: hidden;
56  /*position: absolute;
57  top: 0;*/
58  z-index: 2;
59 }
60
61 .offline .controller {
62  background: rgba(247, 247, 247, .1);
63  height: 100vh;
64  left: 0;
65  /*position: absolute;*/
66  top: 0;
67  width: 100vw;
68  z-index: 1;
69 }
70
71 #offline-resources {
72  display: none;
73 }
74
75 @media (max-width: 420px) {
76  .suggested-left > #control-buttons, .suggested-right > ↩
      #control-buttons {
77   float: none;
78  }
79  .snackbar {
80   left: 0;
81   bottom: 0;
82   width: 100%;
83   border-radius: 0;
84  }
85 }
86
87 @media (max-height: 350px) {
88  h1 {
89   margin: 0 0 15px;
```

```
 }
 .icon-offline {
  margin: 0 0 10px;
 }
 /*.interstitial-wrapper {
  margin-top: 5%;
 }
 .nav-wrapper {
  margin-top: 30px;
 }*/
}

@media (min-width: 600px) and (max-width: 736px) and (orientation: landscape) {
 .offline .interstitial-wrapper {
  margin-left: 0;
  margin-right: 0;
 }
}

@media (min-width: 420px) and (max-width: 736px) and (min-height: 240px) and (max-height: 420px) and (orientation:landscape) {
 /*.interstitial-wrapper {
  margin-bottom: 100px;
 }*/
}

/*@media (min-height: 240px) and (orientation: landscape) {
 .offline .interstitial-wrapper {
  margin-bottom: 90px;
 }
 .icon-offline {
  margin-bottom: 20px;
 }
}*/

@media (max-height: 320px) and (orientation: landscape) {
 .icon-offline {
  margin-bottom: 0;
 }
}

@media (max-width: 240px) {
 .interstitial-wrapper {
  overflow: inherit;
  padding: 0 8px;
 }
}

.materialize-textarea {
 height: 200px !important;
}

textarea {
 max-height: 300px !important;
 overflow-y: scroll !important;
}

.container {
 margin: 0 auto !important;
 max-width: 4000px !important;
 width: 95% !important;
}

```

```
#main-content{
 height: 0px !important;
}

hr {
  display: block;
  height: 1px;
  border: 0;
  border-top: 1px solid #ccc;
  margin: 1em 0;
  padding: 0;
}
```

15.5 function

```
// Simulate a key press
// 38=up, 40=down, 32=space
// "keyup", "keydown"
function simulateKeyPress(keycode, type) {
  var evt = new Event(type);
  evt.keyCode=keycode;
  evt.which=evt.keyCode;
  document.dispatchEvent(evt);
}

// Transform a multidimensionnal array into an 1 dimension array
function ravel(array) {
  var result = new Array();
  if (typeof array[0] == "undefined" || typeof array[0] == "number") {//TO CHANGE
    result = array;
  }
  else {
    for (var i = 0; i < array.length; i++) {
      for (var j = 0; j < array[i].length; j++) {
        result.push(array[i][j]);
      }
    }
  }

  return result;
}
```

15.6 main

```
/*************************************************************
* Author : De Biasi Loris
* Description : The main script of the application
* Version : 0.1
* Date : 24.04.2017
*************************************************************/

// Set the state of the AI
var isRunning = false;

var myGameManager;

// Associates a game to an index
```

```
14 var games = {
15   TREX: 1,
16   FLAPPY: 2
17 }
18
19 var activation;
20 var selectionMethod;
21 var crossoverMethod;
22 var mutationMethod;
23
24 // Store the current game index
25 var currentGameIndex = games.TREX;
26
27 // Game variable
28 var runner;
29
30 // Neural net variable
31 var gen;
32
33 // Resfresh rate of the AI [FPS]
34 var AIResfreshRate = 180;
35
36 // Var to calculate fitness
37 var fitness = 0;
38 var lastValue = 1000;
39
40 // Reference to the fitness chart
41 var fitnessChart;
42
43 // Canvas
44 var c = document.getElementById("neuralNet");
45 var ctx = c.getContext("2d");
46
47 // Get the canvas of the neural net visualisation
48 var neuralNetCanvas = document.getElementById('neuralNet');
49
50 // Var used for the timer
51 var timer = 0;
52 var timePassed = 0;
53
54 // Resfresh rate for the neural net preview
55 var nnPreviewCounter = 0;
56 var nnPreviewRefreshRate = 10; // [FPS]
57
58 // When the page is fully loaded
59 $(document).ready(function(){
60   // Enable the modals and configure them
61   $('.modal').modal({
62     dismissible: true, // Modal can be dismissed by clicking outside ↩
    of the modal
63     opacity: .5, // Opacity of modal background
64     inDuration: 300, // Transition in duration
65     outDuration: 200, // Transition out duration
66     startingTop: '4%', // Starting top style attribute
67     endingTop: '10%', // Ending top style attribute
68     ready: function(modal, trigger) { // Callback for Modal open. ↩
    Modal and trigger parameters available.
69       //console.log(modal, trigger);
70     },
71     complete: function() { // Callback for Modal close
72       //Clear the modals on close
73       clearModals();
74     }
75   });
```

```
76
77    // Configuration of the chart
78    var data = {
79      datasets: [
80        {
81          label: "Best fitness",
82          fill: true,
83          lineTension: 0.1,
84          backgroundColor: "rgba(75,192,192,0.4)",
85          borderColor: "rgba(75,192,192,1)",
86          borderCapStyle: 'butt',
87          borderDash: [],
88          borderDashOffset: 0.0,
89          borderJoinStyle: 'miter',
90          pointBorderColor: "rgba(75,192,192,1)",
91          pointBackgroundColor: "#fff",
92          pointBorderWidth: 1,
93          pointHoverRadius: 5,
94          pointHoverBackgroundColor: "rgba(75,192,192,1)",
95          pointHoverBorderColor: "rgba(220,220,220,1)",
96          pointHoverBorderWidth: 2,
97          pointRadius: 5,
98          pointHitRadius: 10,
99          spanGaps: false,
100       }
101     ]
102   };
103
104   //Creation of the empty chart
105   var chartCtx = document.getElementById("Chart");
106   fitnessChart = new Chart(chartCtx, {
107     type: 'line',
108     data: data,
109     options: {
110       title: {
111         display: true,
112         text: 'Best fitness over generations' // Title
113       }
114     }
115   });
116
117   // Event when a file is loaded
118   document.getElementById('selectedFile').addEventListener('change', ↩
       readSingleFile, false);
119
120   // Call "resizeNeuralNetCanvas" when the window has been resized
121   window.addEventListener('resize', resizeNeuralNetCanvas, false);
122
123   // Enable the selector
124   $(document).ready(function() { $('select').material_select(); });
125
126   // Neural net configuration
127   activation = new sigmoid();
128   selectionMethod = new rouletteWheelSelection();
129   crossoverMethod = new singlePointCrossover();
130   mutationMethod = new mutationWithRate(0.2); // [0..1]
131
132   // Check wich game should be instanciate
133   if (currentGameIndex == games.TREX) {
134     $("select[name=gameSelector] option[value=1]").prop('selected', ↩
       'selected');
135
136     myGameManager = new trexManager();
137   }
```

```
138   else if (currentGameIndex == games.FLAPPY) {
139     $("select[name=gameSelector] option[value=2]").prop('selected', ↩
       'selected');
140
141     myGameManager = new flappyManager();
142   }
143
144   // Instanciate the game
145   runner = myGameManager.instanciateGame();
146
147   // Create a new generation
148   gen = new Generation(myGameManager.defaultTopology, ↩
       myGameManager.defaultNumberOfGenomes, activation, selectionMethod, ↩
       crossoverMethod, mutationMethod);
149
150   // Update the interface
151   updateInterface();
152
153   // Resize the canvas depending the container size
154   resizeNeuralNetCanvas();
155
156   // Print the value of the neural net on the "export" modals
157   $( "#neuralNetSelector" ).change(function() {
158     var nnIndex = $('#neuralNetSelector').val();
159     printSelectedNeuralNet(gen.genomes[nnIndex]);
160   });
161
162   // Event when the game is changed
163   $( "select[name=gameSelector]" ).change(function() {
164     // Check if the selected option is the same as before
165     if (currentGameIndex != $('select[name=gameSelector]').val()) {
166       currentGameIndex = $('select[name=gameSelector]').val();
167
168       // Change the game depending the option selected
169       if (currentGameIndex == games.TREX) {
170         $('.interstitial-wrapper').html('<div id="main-content"><div ↩
             class="icon icon-offline" alt="" style="visibility: ↩
             hidden;"></div></div>');
171         $('.interstitial-wrapper').append('<div ↩
             id="offline-resources"><img id="offline-resources-1x" ↩
             src="assets/default_100_percent/100-offline-sprite.png"> ↩
             <img id="offline-resources-2x" ↩
             src="assets/default_200_percent/200-offline-sprite.png"> ↩
             </div>');
172         myGameManager = new trexManager();
173       }
174       else if (currentGameIndex == games.FLAPPY) {
175         myGameManager = new flappyManager();
176       }
177
178       // Instanciate the game
179       runner = myGameManager.instanciateGame();
180
181       //Create a new generation
182       gen = new Generation(myGameManager.defaultTopology, ↩
             myGameManager.defaultNumberOfGenomes, activation, ↩
             selectionMethod, crossoverMethod, mutationMethod);
183
184       // Draw the neural net
185       gen.drawNeuralNet(c,ctx);
186
187       // Reset the chart
188       removeData(fitnessChart);
189
```

```
190         // Reset the timer
191         timePassed = 0;
192 
193         //Change the running state
194         changeRunningState(false);
195       }
196     });
197 
198     // Timer for the application
199     setInterval(function(){
200       startAI();
201     }, 1000 / AIResfreshRate);
202 });
203 
204 // Change the state of the AI
205 function changeRunningState(state) {
206     if (typeof state == "undefined") {
207       isRunning = !isRunning;
208     }
209     else {
210       isRunning = state;
211     }
212 
213     // Show if the AI is started or stopped
214     if (isRunning == true) {
215       $("#stateBtn").html('STOP<i class="material-icons ↩
            right">power_settings_new</i>');
216       Materialize.toast('AI Started', 4000);
217 
218       // Unpause the game
219       myGameManager.play(runner);
220     }
221     else{
222       $("#stateBtn").html('START<i class="material-icons ↩
            right">power_settings_new</i>');
223       Materialize.toast('AI Stopped', 4000);
224 
225       // Pause the game
226       myGameManager.pause(runner);
227     }
228 }
229 
230 //
231 function startAI() {
232     // Check if the AI should "run"
233     if (isRunning == true) {
234       // Get the normalized data
235       var gameValue = myGameManager.getNormalizedInputValues(runner);
236 
237       // Check if some value were found
238       if (gameValue.length > 0) {
239 
240         // timer
241         timer++;
242         if (timer >= AIResfreshRate) {
243           timePassed++;
244           $("#timeIndex").html(timePassed);
245           timer = 0;
246         }
247 
248         // Neural net preview refresh
249         nnPreviewCounter++;
250         if (nnPreviewCounter >= (AIResfreshRate / ↩
              nnPreviewRefreshRate)) {
```

```
251           // Draw the neural net
252           gen.drawNeuralNet(c,ctx);
253           nnPreviewCounter = 0;
254         }
255
256         // Run the generation
257         var result = gen.run(gameValue);
258
259         // Make a game action
260         myGameManager.action(runner, result);
261
262         // Update the fitness of the AI
263         myGameManager.fitness(runner);
264
265         // When the AI die
266         if (myGameManager.isDead(runner)) {
267           // Restart the game
268           var fitness = myGameManager.tmpFitness;
269           myGameManager.restart(runner);
270
271           // Check if the game has restarted before changing generation
272           if (!myGameManager.isDead(runner)) {
273             // Change the generation and save the fitness
274             gen.nextGen(fitness, fitnessChart);
275
276             // Update the interface
277             updateInterface();
278           }
279         }
280       }
281     }
282 }
283
284 // Remove all the data in a chart
285 function removeData(chart) {
286   chart.data.datasets[0].data = null;
287   chart.data.labels = [];
288   chart.update();
289 }
290
291 // Get the best neural net index
292 function getBestNN(){
293   var maxFitness = -1;
294   var bestNNIndex;
295
296   for (var i = 0; i < gen.genomes.length; i++) {
297     if (gen.genomes[i].fitness > maxFitness) {
298       maxFitness = gen.genomes[i].fitness;
299       bestNNIndex = i;
300     }
301   }
302
303   printSelectedNeuralNet(gen.genomes[bestNNIndex]);
304 }
305
306 // Print the weights of a neural net
307 function printSelectedNeuralNet(bestGenome) {
308   var weights = bestGenome.getWeights();
309   $("#textareaExport").html(weights.toString());
310 }
311
312 // Refresh the neural net index selector (export modals)
313 function resfreshNNselection() {
314   $("#neuralNetSelector").empty();
```

```
315   for (var i = 0; i < gen.genomes.length; i++) {
316     $("#neuralNetSelector").append('<option value="'+i+'">Neural net ↩
          '+(i+1)+'</option>');
317   }
318
319   $('select').material_select();
320
321   var nnIndex = $('#neuralNetSelector').val();
322   printSelectedNeuralNet(gen.genomes[nnIndex]);
323 }
324
325 // Update the interface
326 function updateInterface() {
327   $('#generationIndex').html(gen.generation + 1);
328   $('#genomeIndex').html(gen.currentGenome + 1 + "/" + ↩
        gen.genomes.length);
329 }
330
331 // Resize the canvas depending de windows size
332 function resizeNeuralNetCanvas() {
333   neuralNetCanvas.width = $("#neuralNetPreview").width();
334   neuralNetCanvas.height = $("#neuralNetPreview").height();
335
336   // Redraw the neural net after changing the size
337   gen.drawNeuralNet(c,ctx);
338 }
339
340 // Add a fitness to a chart
341 function addDataToChart(chart,fitness) {
342   var chartLength = chart.data.datasets[0].data.length;
343
344   chart.data.datasets[0].data.push(fitness);
345   chart.data.labels.push("Gen " + (chartLength + 1));
346   chart.update();
347 }
348
349 // Download the neural network (exported neural network)
350 function downloadFile(filename, text) {
351   var element = document.createElement('a');
352   element.setAttribute('href', 'data:text/plain;charset=utf-8,' + ↩
       encodeURIComponent(text));
353   element.setAttribute('download', filename);
354
355   element.style.display = 'none';
356   document.body.appendChild(element);
357   element.click();
358   document.body.removeChild(element);
359
360   Materialize.toast('Neural network succesfully exported', 4000);
361 }
362
363 // Read a file and print it in a textarea
364 function readSingleFile(evt) {
365   //Retrieve the first (and only!) File from the FileList object
366   var file = evt.target.files[0];
367
368   // If the file exist/is selected
369   if (file) {
370     var reader = new FileReader();
371
372     // Load the file
373     reader.onload = function(e) {
374       // Get the file data
375       var contents = e.target.result;
```

```
376
377         // Add the content to the textarea
378         $("#textareaImport").val(contents);
379       }
380
381       // Read the text file
382       reader.readAsText(file);
383     } else {
384       Materialize.toast('Failed to load file', 4000);
385     }
386 }
387
388 // Clear the modals
389 function clearModals() {
390   $('.modal').find('input:text, input:password').val('');
391   $('.modal').find('input:radio, input:checkbox').prop('checked', ↩
      false);
392 }
393
394 // Import a neural network
395 function importNeuralNetwork(){
396   gen = new Generation(myGameManager.defaultTopology, ↩
        myGameManager.defaultNumberOfGenomes, activation, ↩
        selectionMethod, crossoverMethod, mutationMethod);
397
398   var genToImport = JSON.parse("[" + $("#textareaImport").val() + "]");
399   var oldGen = gen.genomes[0].getWeights();
400
401   if (genToImport.length != oldGen.length) {
402     Materialize.toast('Importation error, missing or too many data ↩
        (wrong game ?)', 6000)
403   }
404   else {
405     gen.genomes[0].setWeights(genToImport);
406
407     Materialize.toast('Neural network succesfully imported', 4000)
408
409     // Reset everything
410     removeData(fitnessChart);
411     myGameManager.restart(runner);
412     timePassed = 0;
413     $("#timeIndex").html(timePassed);
414     updateInterface()
415     gen.drawNeuralNet(c,ctx);
416   }
417
418   $('#importModal').modal('close');
419 }
420
421 // For debug (faster to type)
422 function log(msg) {
423   console.log(msg);
424 }
```

15.7 gameManager

```
1 /*************************************************************
2 * Author : De Biasi Loris
3 * Description : The game manager class. Allows to add
4 *       games easily
5 * Version : 0.1
```

```
6  * Date : 22.05.2017
7  **********************************************************/
8
9  /** Abstract class for game manager */
10 class gameManager {
11   constructor() {
12     if (new.target === gameManager) {
13       throw new TypeError("Cannot construct gameManager instances directly");
14     }
15   }
16
17   instanciateGame() { throw new Error("Must override method"); }
18   getNormalizedInputValues(game) { throw new Error("Must override method"); }
19   action(game, result) { throw new Error("Must override method"); }
20   fitness(game) { throw new Error("Must override method"); }
21   isDead(game) { throw new Error("Must override method"); }
22   play(game) { throw new Error("Must override method"); }
23   pause(game) { throw new Error("Must override method"); }
24   restart(game) { throw new Error("Must override method"); }
25 }
26
27 /**
28  * Trex game manager
29  * @extends gameManager
30  */
31 class trexManager extends gameManager {
32   /**
33    * Create a trex manager.
34    * @constructor
35    */
36   constructor() {
37     // Parent constructor
38     super();
39
40     this.defaultTopology = [1,1,1];
41     this.defaultNumberOfGenomes = 12;
42     this.tmpFitness = 0;
43     this.lastXpos = 1000;
44   }
45
46   /**
47    * Create a trex manager.
48    * @return {game} The game
49    */
50   instanciateGame() {
51     return new Runner('.interstitial-wrapper');
52   }
53
54   /**
55    * Get the normalized input values
56    * @param {game} game - The game
57    * @return {number} The input
58    */
59   getNormalizedInputValues(game) {
60     var inputs = [];
61
62     if (typeof game.horizon.obstacles[0] != "undefined") {
63       var maxVelocity = 13;
64       var maxDistance = 600 + 25;
65       var maxYPosition = 105;
66       var maxSize = 3;
67
```

```
68        var normalizedVelocity = game.currentSpeed.toFixed(3) / ↩
              maxVelocity;
69        var normalizedDistance = game.horizon.obstacles[0].xPos / ↩
           maxDistance;
70        var normalizedYPosition = game.horizon.obstacles[0].yPos / ↩
           maxYPosition;
71        var normalizedSize = game.horizon.obstacles[0].size / maxSize;
72
73        inputs = ↩
           [normalizedDistance];//,normalizedYPosition,normalizedSize ↩
           ,normalizedVelocity
74      }
75
76      return inputs;
77    }
78
79    /**
80     * Make an action depending the result
81     * @param {game} game - The game
82     * @param {game} result - The result
83     */
84    action(game, result) {
85      // Make action depending the neural net output
86      if (result > 0.6) { // greater than 0.6 [press up]
87        simulateKeyPress(38, "keydown");
88      }
89      else if (result < 0.4) { // less than 0.4 [press down]
90        simulateKeyPress(40, "keydown");
91      }
92      else {
93        //do nothing
94      }
95    }
96
97    /**
98     * Update the fitness
99     * @param {game} game - The game
100    */
101   fitness(game) {
102     if (game.horizon.obstacles[0].xPos > this.lastXpos) {
103       this.tmpFitness++;
104     }
105     this.lastXpos = game.horizon.obstacles[0].xPos;
106   }
107
108   /**
109    * Check if the player is dead
110    * @param {game} game - The game
111    * @return {boolean} The status of the game
112    */
113   isDead(game) {
114     return game.crashed;
115   }
116
117   /**
118    * Unpause the game
119    * @param {game} game - The game
120    */
121   play(game) {
122     // Unpause the trex game
123     game.play();
124     // Simulate key press to start the game
125     simulateKeyPress(38, "keydown");
126   }
```

```
127
128   /**
129    * Pause the game
130    * @param {game} game - The game
131    */
132   pause(game) {
133     game.stop();
134   }
135
136   /**
137    * Restart the game
138    * @param {game} game - The game
139    */
140   restart(game) {
141     this.tmpFitness = 0;
142     this.lastXpos = 1000;
143     game.restart();
144   }
145 }
146
147 /**
148  * Flappy game manager
149  * @extends gameManager
150  */
151 class flappyManager extends gameManager {
152   /**
153    * Create a flappy manager.
154    * @constructor
155    */
156   constructor() {
157     // Parent constructor
158     super();
159
160     this.defaultTopology = [2,2,1];
161     this.defaultNumberOfGenomes = 60;
162     this.tmpFitness = 0;
163   }
164
165   /**
166    * Create a trex manager.
167    * @return {game} The game
168    */
169   instanciateGame() {
170     return new flappyBird();
171   }
172
173   /**
174    * Get the normalized input values
175    * @param {game} game - The game
176    * @return {number} The input
177    */
178   getNormalizedInputValues(game) {
179     var inputs = [];
180
181     if (typeof game.game.backgroundSpeed != "undefined") {
182       if (game.game.pipes.length > 0) {
183         var normalizedY = game.game.birds[0].y / game.game.height;
184         var normalizedPipe;
185
186         var nextHoll = 0;
187         for(var i = 0; i < game.game.pipes.length; i+=2){
188           if(game.game.pipes[i].x + game.game.pipes[i].width > ↩
                game.game.birds[0].x){
189             nextHoll = game.game.pipes[i].height/game.game.height;
```

```
190             break;
191           }
192         }
193
194         normalizedPipe = nextHoll;
195
196         // Create the input array for the neural network
197         inputs = [normalizedY,normalizedPipe];
198       }
199     }
200
201     return inputs;
202   }
203
204   /**
205    * Make an action depending the result
206    * @param {game} game - The game
207    * @param {game} result - The result
208    */
209   action(game, result) {
210     // Make action depending the neural net output
211     if (result > 0.5) { // greater than 0.5 [press up]
212       game.game.birds[0].flap();
213     }
214   }
215
216   /**
217    * Update the fitness
218    * @param {game} game - The game
219    */
220   fitness(game) {
221     this.tmpFitness = game.game.score;
222   }
223
224   /**
225    * Check if the player is dead
226    * @param {game} game - The game
227    * @return {boolean} The status of the game
228    */
229   isDead(game) {
230     return game.game.isItEnd();
231   }
232
233   /**
234    * Unpause the game
235    * @param {game} game - The game
236    */
237   play(game) {
238     game.game.run = true;
239   }
240
241   /**
242    * Pause the game
243    * @param {game} game - The game
244    */
245   pause(game) {
246     game.game.run = false;
247   }
248
249   /**
250    * Restart the game
251    * @param {game} game - The game
252    */
253   restart(game) {
```

```
254     this.tmpFitness = 0;
255     game.game.start();
256   }
257 }
```

15.8 generation

```
1  /*************************************************************
2  * Author : De Biasi Loris
3  * Description : The generation class.
4  * Version : 0.1
5  * Date : 24.04.2017
6  *************************************************************/
7
8  /** Class used to create a generation. */
9  class Generation {
10   /**
11    * Create a generation.
12    * @constructor
13    * @param {number} pTopology - The topology of the neural network
14    * @param {number} pNbOfGenome - The number of genomes
15    * @param {activationFunction} pActivationFunction - The activation ↩
        function
16    * @param {selectionMethod} pSelectionMethod - The selection method
17    * @param {crossoverMethod} pCrossoverMethod - The crossover method
18    * @param {mutationMethod} pMutationMethod - The mutation method
19    */
20   constructor(pTopology, pNbOfGenome, pActivationFunction, ↩
       pSelectionMethod, pCrossoverMethod, pMutationMethod) {
21     this.genomes = [];
22     if (pNbOfGenome < 4) {
23       this.numberOfGenomes = 4;
24     }
25     else {
26       this.numberOfGenomes = pNbOfGenome;
27     }
28     this.generation = 0;
29     this.currentGenome = 0;
30
31     // Store the power for the fitness
32     this.power = 1;
33
34     // Set the different method used to create a new generation
35     this.selection = pSelectionMethod;
36     this.crossover = pCrossoverMethod;
37     this.mutation = pMutationMethod;
38
39     // Forced to store them to create a new object from the object ↩
         himself
40     this.topology = pTopology;
41     this.activationFunction = pActivationFunction;
42
43     this.bestNeuralNet = new Genome(pTopology, pActivationFunction);
44
45     // Create the new generation
46     for (var i = 0; i < this.numberOfGenomes; i++) {
47       this.genomes.push(new Genome(pTopology, pActivationFunction));
48     }
49   }
50
51   /**
```

```
52      * Run the current generation and return the output value
53      * @param {number} input - The input values
54      * @return {number} the output value.
55      */
56    run(input) {
57      // Feed the neural network with the value
58      this.genomes[this.currentGenome].feedForward(input);
59
60      // Check which move the AI should do
61      var result = this.genomes[this.currentGenome].getOutput();
62
63      return result;
64    }
65
66    /**
67      * Create the next generation
68      * @param {number} pFitness - The fitness
69      * @param {chart} pFitnessChart - The fitness chart
70      */
71    nextGen(pFitness, pFitnessChart) {
72      // Store the score of the current genome
73      this.genomes[this.currentGenome].setFitness(pFitness**this.power);
74
75      // Get the best neural net ever made
76      if (this.bestNeuralNet.fitness < pFitness**this.power) {
77        this.bestNeuralNet = this.genomes[this.currentGenome];
78      }
79
80      // Change the generation if we used all the genomes
81      if (this.currentGenome >= this.genomes.length - 1) {
82        // Reset the current genome number
83        this.currentGenome = 0;
84
85        // Change the generation
86        this.generation++;
87
88        // Add the data to the chart
89        addDataToChart(pFitnessChart,this.getBestFitness());
90
91        // Create a new generation
92        var selected = this.selection.process(this);
93        var newGen = this.crossover.process(this,selected);
94        var newMutatedGen = this.mutation.process(newGen);
95        this.genomes = newMutatedGen;
96        //this.mutation.rate =
97      }
98      else {
99        this.currentGenome++;
100     }
101   }
102
103   /**
104     * Get the fitness average of the generation
105     * @return {number} the average value.
106     */
107   getFitnessAverage() {
108     var fitnessAverage = 0;
109     for (var i = 0; i < this.genomes.length; i++) {
110       fitnessAverage += this.genomes[i].fitness;
111     }
112     fitnessAverage /= this.genomes.length;
113
114     return Math.pow(fitnessAverage,1/this.power);
115   }
```

```
116
117   /**
118    * Get the best fitness of the generation
119    * @return {number} the biggest value.
120    */
121   getBestFitness() {
122     var bestFitness = 0;
123     for (var i = 0; i < this.genomes.length; i++) {
124       if (bestFitness < this.genomes[i].fitness) {
125         bestFitness = this.genomes[i].fitness;
126       }
127     }
128
129     return Math.pow(bestFitness,1/this.power);
130   }
131
132   /**
133    * Set the weights of each genomes
134    * @return {number} All the weights.
135    */
136   getAllWeights(){
137     var result = [];
138
139     for (var i = 0; i < this.genomes.length; i++) {
140       result.push(this.genomes[i].getWeights());
141     }
142
143     return result
144   }
145
146   /**
147    * Set the weights of each genomes
148    * @param {number} pWeights - All the weights
149    */
150   setAllWeights(pWeights){
151     for (var i = 0; i < this.genomes.length; i++) {
152       this.genomes[i].setWeights(pWeights[i]);
153     }
154   }
155
156   /**
157    * Draw the neural network
158    * @param {canvas} canvas - The canvas
159    * @param {context} context - The context of the canvas
160    */
161   drawNeuralNet(canvas,context) {
162     this.genomes[this.currentGenome].drawNeuralNet(canvas,context);
163   }
164 }
```

15.9 genome

```
1 /*************************************************************
2 * Author : De Biasi Loris
3 * Description : The genome class
4 * Version : 0.1
5 * Date : 24.04.2017
6 *************************************************************/
7
8 /** Class used to instanciate a genome. Used to store the fitness of ↩
    a neural network */
```

```
9  class Genome {
10   /**
11    * Create a genome.
12    * @constructor
13    * @param {number} pTopology - The topology of the neural network
14    * @param {activationFunction} pActivationFunction - The activation ↩
          function
15    */
16   constructor(pTopology, pActivationFunction) {
17     this.fitness = 0;
18     this.neuralNet = new NeuralNetwork(pTopology, pActivationFunction);
19   }
20
21   /**
22    * Set the fitness of this genome
23    * @param {number} pFitness - The fitness
24    */
25   setFitness(pFitness) {
26     this.fitness = pFitness;
27   }
28
29   /**
30    * Get the fitness of this genome
31    * @return {number} the weights of the genome.
32    */
33   getWeights() {
34     return this.neuralNet.getWeights();
35   }
36
37   /**
38    * Set the weight of this genome
39    * @param {number} pWeights - The weights of the genome
40    */
41   setWeights(pWeights) {
42     this.neuralNet.setWeights(pWeights);
43   }
44
45   /**
46    * Feed forward the input values
47    * @param {number} inputValues - The input values
48    */
49   feedForward(inputValues) {
50     this.neuralNet.feedForward(inputValues);
51   }
52
53   /**
54    * Get the output of the neural network
55    * @return {number} the output value.
56    */
57   getOutput() {
58     return this.neuralNet.getOutput();
59   }
60
61   /**
62    * Draw the neural network
63    * @param {canvas} canvas - The canvas
64    * @param {context} context - The context of the canvas
65    */
66   drawNeuralNet(canvas,context) {
67     this.neuralNet.drawNeuralNet(canvas,context);
68   }
69 }
```

15.10 neural_network

```
/*************************************************************
* Author : De Biasi Loris
* Description : The neural network class
* Version : 0.1
* Date : 24.04.2017
*************************************************************/

/** Class used to instanciate a neural network */
class NeuralNetwork {
  /**
   * Create a neural network.
   * @constructor
   * @param {number} pTopology - The topology of the neural network
   * @param {activationFunction} pActivationFunction - The activation function
   */
  constructor(pTopology, pActivationFunction) {
    // Store the number of layer
    this.numberOfLayers = pTopology.length;

    // Store the layers
    this.layers = [];

    // Create the layers structure
    for (var layerNum = 0; layerNum < this.numberOfLayers; layerNum++) {
      var numberOfOutputs = 0
      // If the current layer num is not the last one
      if (layerNum != pTopology.length - 1) {
        // Store the size of the next layer
        numberOfOutputs = pTopology[layerNum + 1];
      }

      // Create a new layer
      this.layers.push(new Layer(pTopology[layerNum], numberOfOutputs, pActivationFunction));
    }

    //
    this.layers[this.layers.length - 1].neurons[this.layers[this.layers.length - 1].neurons.length - 1].outputValue = 1;
  }

  /**
   * Feed forward the values
   * @param {number} inputValues - The input values
   */
  feedForward(inputValues) {
    // Assign the input values into the input neurons
    for (var i = 0; i < inputValues.length; i++) {
      this.layers[0].neurons[i].outputValue = inputValues[i];
    }

    // Forward propagate (start from 1 because input layer is set before)
    for (var layerNum = 1; layerNum < this.layers.length; layerNum++) {
      // Store the previous layer
      var previousLayer = this.layers[layerNum - 1];

      // feed forward all the values
```

```
56        for (var neuronNum = 0; neuronNum < this.layers[layerNum].neurons.length - this.layers[layerNum].numberOfBias; neuronNum++) {
57          this.layers[layerNum].neurons[neuronNum].feedForward(previousLayer);
58        }
59      }
60    }
61
62    /**
63     * Get the weights of this neural network
64     * @return {number} The weights of the neural network.
65     */
66    getWeights() {
67      var weights = [];
68      for (var i = 0; i < this.layers.length; i++) {
69        weights.push(this.layers[i].getWeights());
70      }
71      return ravel(weights);
72    }
73
74    /**
75     * Set the weights of this neural network
76     */
77    setWeights(pWeights) {
78      var pos = 0;
79      for (var i = 0; i < this.numberOfLayers; i++) {
80        var nextLayerLength = 1;
81        var nbBias = this.layers[i].numberOfBias;
82
83        // If this isn't the last layer
84        if (i+1 < this.layers.length) {//typeof this.layers[i+1] != "undefined"
85          nbBias = this.layers[i+1].numberOfBias;
86          nextLayerLength = this.layers[i+1].neurons.length;
87        }
88
89        //Remove the number of bias from the next layer size
90        nextLayerLength -= nbBias;
91
92        //Get the number of neurons in this layer
93        var nbOfNeuronsInLayer = (this.layers[i].neurons.length) * nextLayerLength;
94
95        //Slice the array and send the right piece at the right layer
96        this.layers[i].setWeights(pWeights.slice(pos, pos + nbOfNeuronsInLayer), nextLayerLength);
97
98        //Increment the pos
99        pos+=nbOfNeuronsInLayer;
100     }
101   }
102
103   /**
104    * Get the result of the output neuron
105    * @return {number} The output value.
106    */
107   getOutput() {
108     return this.layers[this.layers.length - 1].neurons[0].outputValue;
109   }
110
111   /**
112    * Draw the neural network
113    */
```

```
114   drawNeuralNet(canvas,context) {
115     // The size in pixel for a neuron
116     var neuronSize = 0;
117     var neuronSpace = 10;
118     var spaceFromBorder = 20;
119     var pourcent = 15;
120
121     // Get the biggest number of neuron per layer
122     var maxNeuron = 0;
123     for (var i = 0; i < this.layers.length; i++) {
124       if (this.layers[i].neurons.length > maxNeuron) {
125         maxNeuron = this.layers[i].neurons.length;
126       }
127     }
128
129     // Get the lowest size (used to create circle)
130     if ((canvas.height / maxNeuron) < (canvas.width / ↩
          this.layers.length)) {
131       neuronSize = (canvas.height / maxNeuron);
132     }
133     else {
134       neuronSize = canvas.width / this.layers.length;
135     }
136
137     //
138     neuronSize -= neuronSpace;
139
140     context.clearRect(0,0,canvas.width,canvas.height);
141
142     //
143     var spaceBetweenNeuronX = 0;
144     var spaceBetweenNeuronY = 0;
145     for (var y = 0; y < this.layers.length; y++) {
146       // Get the size between neuron [Y axis]
147       spaceBetweenNeuronY = ((canvas.width - (spaceFromBorder * 2)) / ↩
            (this.layers.length - 1)) - neuronSize / 2;
148       for (var x = 0; x < this.layers[y].neurons.length; x++) {
149         // Get the size between neuron [X axis]
150         spaceBetweenNeuronX = ((canvas.height - ↩
              (this.layers[y].neurons.length * neuronSize)) / ↩
              (this.layers[y].neurons.length + 1));
151
152         var posY = spaceBetweenNeuronY * y + spaceFromBorder + ↩
              neuronSize / 2;
153         var posX = spaceBetweenNeuronX * (x+1) + neuronSize * x + ↩
              neuronSize / 2;
154
155         // Change the color if this is the bias
156         if (x == this.layers[y].neurons.length - 1) {
157           context.fillStyle="#BDBDBD";
158           context.strokeStyle="#BDBDBD";
159         }
160         else {
161           context.fillStyle="#ee6e73";
162           context.strokeStyle="#ee6e73";
163         }
164
165         context.lineWidth=1.5;
166
167         // Draw the circle
168         context.beginPath();
169         context.arc(posY, posX, neuronSize / 2, 0, 2 * Math.PI);
170         context.stroke();
171         context.closePath();
```

```
172
173         context.font = "20px Arial";
174
175         // Draw the output values of each neuron
176         context.fillText(this.layers[y].neurons[x].outputValue
               .toFixed(3),posY - (context.measureText(this.layers[y]
               .neurons[x].outputValue.toFixed(3)).width / 2), posX + 10);
177
178         // Check if this layer is the last one
179         if (y + 1 < this.layers.length) {
180           var nextSpaceBetweenNeuronX = (canvas.height -
                (this.layers[y+1].neurons.length * neuronSize)) /
                (this.layers[y+1].neurons.length + 1);
181           for (var neuronIndex = 0; neuronIndex <
                this.layers[y+1].neurons.length - 1; neuronIndex++) {
182
183             // Get the line position
184             var linePosY = posY + neuronSize / 2;
185             var nextPosY = spaceBetweenNeuronY * (y+1) +
                   spaceFromBorder;
186             var nextPosX = nextSpaceBetweenNeuronX * (neuronIndex+1)
                   + neuronSize * neuronIndex + neuronSize / 2;
187
188             var yPourcent = Math.ceil(Math.ceil(linePosY - nextPosY)
                   / 100 * pourcent);
189             var xPourcent = Math.ceil(Math.ceil(posX - nextPosX) /
                   100 * pourcent);
190
191             context.beginPath();
192
193             // Draw the line
194             context.moveTo(linePosY, posX);
195             context.lineTo(nextPosY, nextPosX);
196             context.stroke();
197
198             // Change the font
199             context.font = "12px Arial";
200
201             // Draw a rectangle behind the text
202             context.fillStyle="#FFFFFF";
203             context.fillRect(linePosY - yPourcent - 2, posX -
                   xPourcent - 11, 38, 13);
204
205             context.fillStyle="#000000";
206
207             // Draw the weight of each neuron
208             context.fillText(this.layers[y].neurons[x].
                   outputWeights[neuronIndex].weight.toFixed(3),linePosY
                   - yPourcent, posX - xPourcent);
209           }
210         }
211       }
212     }
213   }
214 }
```

15.11 layer

```
1 /*************************************************************
2 * Author : De Biasi Loris
3 * Description : The layer class
```

```
4  * Version : 0.1
5  * Date : 24.04.2017
6  ************************************************************/
7
8  /** Class used to create a layer for a neural network */
9  class Layer {
10   /**
11    * Create a layer.
12    * @constructor
13    * @param {number} pNbOfNeurons - The number of neurons
14    * @param {number} pNumberOfOutputs - The number of outputs
15    * @param {activationFunction} pActivationFunction - The activation ↩
          function
16    */
17   constructor(pNbOfNeurons, pNumberOfOutputs, pActivationFunction) {
18     // Store the neurons
19     this.neurons = [];
20
21     // Store the number of bias
22     this.numberOfBias = 1;
23
24     // Create all the neurons and the bias
25     for (var neuronNum = 0; neuronNum < pNbOfNeurons + ↩
           this.numberOfBias; neuronNum++) {
26       this.neurons.push(new Neuron(pNumberOfOutputs, neuronNum, ↩
           pActivationFunction));
27     }
28   }
29
30   /**
31    * Get the weights of this layer
32    * @return {number} The weights of the layer.
33    */
34   getWeights() {
35     var weights = [];
36     for (var i = 0; i < this.neurons.length; i++) {//- this.numberOfBias
37       weights.push(this.neurons[i].getWeights());
38     }
39     return ravel(weights);
40   }
41
42   /**
43    * Set the weights of this layer
44    */
45   setWeights(pWeights, pNextLayerSize) {
46     var pos = 0;
47     for (var i = 0; i < this.neurons.length; i++) {
48       this.neurons[i].setWeights(pWeights.slice(pos, pos + ↩
           pNextLayerSize));
49       pos+=pNextLayerSize;
50     }
51   }
52 }
```

15.12 neuron

```
1  /************************************************************
2  * Author : De Biasi Loris
3  * Description : The neuron class
4  * Version : 0.1
5  * Date : 24.04.2017
```

```
6  ************************************************************/
7
8  /** Class used to create a neuron for a neural network */
9  class Neuron {
10   /**
11    * Create a neuron.
12    * @constructor
13    * @param {number} pNumberOfOutputs - The number of outputs
14    * @param {number} pIndex - The index of this neuron
15    * @param {activationFunction} pActivationFunction - The activation ↩
          function
16    */
17   constructor(pNumberOfOutputs, pIndex, pActivationFunction) {
18     // Store the number of output
19     this.numberOfOutputs = pNumberOfOutputs;
20
21     // Store the output value
22     this.outputValue = 1;
23
24     // Store the connections
25     this.outputWeights = [];
26
27     // Create the connections
28     for (var c = 0; c < this.numberOfOutputs; c++) {
29       this.outputWeights.push(new Connection());
30     }
31
32     // Store the index of this neuron
33     this.index = pIndex;
34   }
35
36   /**
37    * Get the weights of this neuron
38    * @return {number} The weigthts of this neuron.
39    */
40   getWeights() {
41     var weights = [];
42     for (var i = 0; i < this.outputWeights.length; i++) {
43       weights.push(this.outputWeights[i].getWeight());
44     }
45     return ravel(weights);
46   }
47
48   /**
49    * Set the weights of this neuron
50    * @param {number} pWeights - The weights to set
51    */
52   setWeights(pWeights) {
53     for (var c = 0; c < this.numberOfOutputs; c++) {
54       this.outputWeights[c].setWeight(pWeights[c]);
55     }
56   }
57
58   /**
59    * Feed forward the values
60    * @param {number} previousLayer - The previous layer
61    */
62   feedForward(previousLayer) {
63     var sum = 0;
64
65     //
66     for (var n = 0; n < previousLayer.neurons.length - 1; n++) {
67       sum += previousLayer.neurons[n].outputValue * ↩
             previousLayer.neurons[n].outputWeights[this.index].weight;
```

```
68      }
69
70      this.outputValue = this.activationFunction.normal(sum);
71    }
72  }
```

15.13 connection

```
1   /*************************************************************
2   * Author : De Biasi Loris
3   * Description : The connection class
4   * Version : 0.1
5   * Date : 24.04.2017
6   *************************************************************/
7
8   /** Class used to store the connection weight of a neuron */
9   class Connection {
10    /**
11     * Create a connection.
12     * @constructor
13     */
14    constructor() {
15      // Create the initial weight
16      this.weight = this.randomWeight(-1,1);
17
18      //this.deltaWeight;
19    }
20
21    /**
22     * Get the weight
23     * @return {number} The weights if this connection
24     */
25    getWeight() {
26      return this.weight;
27    }
28
29    /**
30     * Set the weight
31     */
32    setWeight(pWeight) {
33      this.weight = pWeight;
34    }
35
36    /**
37     * Return a random float between min an max (both include)
38     * Source : https://developer.mozilla.org/en-US/docs/ ↩
          Web/JavaScript/Reference/Global_Objects/Math/random
39     * @param {number} min - The min value.
40     * @param {number} max - The max value.
41     * @return {number} The random number
42     */
43    randomWeight(min, max){
44      min = Math.ceil(min);
45      max = Math.floor(max);
46      return Math.random() * (max - min) + min;
47    }
48  }
```

15.14 activation_function

```
/*************************************************************
* Author : De Biasi Loris
* Description : The activation function class
* Version : 0.1
* Date : 24.04.2017
*************************************************************/

/** Abstract class for activation function */
class activationFunction {
  constructor() {
    if (new.target === activationFunction) {
      throw new TypeError("Cannot construct activationFunction instances directly");
    }
  }

  normal(x) { throw new Error("Must override method"); }
  derivative(x) { throw new Error("Must override method"); }
}

/**
 * Sigmoid activation function class
 * @extends activationFunction
 */
class sigmoid extends activationFunction {
  /**
   * Create a sigmoid
   * @constructor
   */
  constructor() {
    // Parent constructor
    super();
  }

  /**
   * Normal function
   * @return {number} The result of the function
   */
  normal(x) {
    return 1/(1+Math.pow(Math.E, -x));
  }

  /**
   * Derivative function
   * @return {number} The result of the derivative function
   */
  derivative(x) {
    return Math.exp(-x / ((1+Math.exp(-x))**2));
  }
}

/**
 * Tanh activation function class
 * @extends activationFunction
 */
class tanh extends activationFunction{
  /**
   * Create a tanh
   * @constructor
   */
  constructor() {
```

```
61      // Parent constructor
62      super();
63    }
64
65    /**
66     * Normal function
67     * @return {number} The result of the function
68     */
69    normal(x) {
70      return Math.tanh(x);
71    }
72
73    /**
74     * Derivative function
75     * @return {number} The result of the derivative function
76     */
77    derivative(x) {
78      return 1 - (x * x);
79    }
80  }
```

15.15 selection

```
1   /*************************************************************
2   * Author : De Biasi Loris
3   * Description : The selection method class
4   * Version : 0.1
5   * Date : 16.06.2017
6   *************************************************************/
7
8   /** Abstract class for selection method */
9   class selection {
10    constructor() {
11      if (new.target === selection) {
12        throw new TypeError("Cannot construct selection instances ↩
              directly");
13      }
14    }
15
16    process(pGeneration) { throw new Error("Must override method"); }
17  }
18
19  /**
20   * NOT USED AND MUST BE CHANGED
21   * @extends selection
22   */
23  class selectionOfBest extends selection {
24    constructor() {
25      // Parent constructor
26      super();
27    }
28
29    /**
30     * Process the selection method
31     * @param {number} pGeneration - The generation.
32     * @return {number} The selected parents.
33     */
34    process(pGeneration) {
35      // Store the best genome
36      var selected = [];
37      var tmpWeight = [];
```

```
38      var genomeToAppend;
39      var max = -1;
40      var genClone;
41      var indexToRemove;
42
43      // Create a clone of the genomes
44      genClone = new Generation(pGeneration.topology, ↩
            pGeneration.numberOfGenomes, pGeneration.activationFunction, ↩
            pGeneration.rate);
45
46      // Get all the weights of the generation
47      for (var g = 0; g < pGeneration.genomes.length; g++) {
48        tmpWeight.push(pGeneration.genomes[g].getWeights());
49      }
50
51      // Set all the weights to the tmp generation (copy)
52      for (var g = 0; g < pGeneration.genomes.length; g++) {
53        genClone.genomes[g].setWeights(tmpWeight[g]);
54      }
55      //
56
57      // Set the fitness
58      for (var i = 0; i < pGeneration.genomes.length; i++) {
59        genClone.genomes[i].setFitness(pGeneration.genomes[i].fitness);
60      }
61
62      // Select the best genomes [2 * (sqrt(nbOfGenomes) / 2)]
63      for (var i = 0; i < ↩
            2*Math.floor(Math.sqrt(pGeneration.genomes.length)/2); i++) { ↩
            // Number of wanted genomes for breeding
64        max = -1;
65        // Check the best fitness
66        for (var j = 0; j < genClone.genomes.length; j++) {
67          // If a genome as a higher score
68          if (max < genClone.genomes[j].fitness) {
69            max = genClone.genomes[j].fitness;
70
71            // Store the best genome
72            genomeToAppend = genClone.genomes[j];
73            indexToRemove = j;
74          }
75        }
76        selected.push(genomeToAppend);
77
78        // Remove the genome from the copied array
79        genClone.genomes.splice(indexToRemove, 1);
80      }
81
82      return selected;
83    }
84  }
85
86  /**
87   * Roulette wheel selection method
88   * @extends selection
89   */
90  class rouletteWheelSelection extends selection{
91    constructor() {
92      // Parent constructor
93      super();
94    }
95
96    /**
97     * Process the selection method
```

```
98      * @param {number} pGeneration - The generation.
99      * @return {number} The selected parents.
100     */
101   process(pGeneration) {
102     var selected = [];
103     var numberOfWantedParents = ↩
          2;//2*Math.floor(Math.sqrt(pGeneration.genomes.length)/2)
104
105     for (var i = 0; i < numberOfWantedParents; i++) { // Number of ↩
          wanted genomes for breeding
106       var sum = 0;
107       var weightSum = 0;
108
109       // Get the total fitness
110       for (var genomeIndex = 0; genomeIndex < ↩
            pGeneration.genomes.length; genomeIndex++) {
111         weightSum += pGeneration.genomes[genomeIndex].fitness + 0.1; ↩
              // +0.1 because some can have 0 fitness
112       }
113
114       // Get a random number between 0 and the weight sum
115       var threshold = Math.random() * weightSum;
116
117       // Go through each genomes
118       for (var genomeIndex = 0; genomeIndex < ↩
            pGeneration.genomes.length; genomeIndex++) {
119         sum += pGeneration.genomes[genomeIndex].fitness + 0.1;
120         // If the sum is bigger than the threshold, the current ↩
              genome is selected and removed from the array
121         if (sum > threshold) {
122           selected.push(pGeneration.genomes[genomeIndex]);
123           pGeneration.genomes.splice(genomeIndex, 1)
124           break;
125         }
126       }
127     }
128
129     return selected;
130   }
131 }
```

15.16 crossover

```
1  /*************************************************************
2  * Author : De Biasi Loris
3  * Description : The crossover method class
4  * Version : 0.1
5  * Date : 16.06.2017
6  *************************************************************/
7
8  /** Abstract class for crossover method */
9  class crossover {
10   constructor() {
11     if (new.target === crossover) {
12       throw new TypeError("Cannot construct crossover instances ↩
           directly");
13     }
14   }
15
16   process(pGeneration,pSelectedGenomes) { throw new Error("Must ↩
       override method"); }
```

```
}

/**
 * Single point crossover method
 * @extends crossover
 */
class singlePointCrossover extends crossover {
  constructor() {
    // Parent constructor
    super();
  }

  /**
   * Process the crossover method
   * @param {number} pGeneration - The generation.
   * @param {number} pSelectedGenomes - The selected genomes.
   * @return {genomes} The modified genomes.
   */
  process(pGeneration,pSelectedGenomes) {
    var children = [];
    // Create a copy of the genomes
    var genClone = new Generation(pGeneration.topology, pGeneration.numberOfGenomes, pGeneration.activationFunction, pGeneration.rate);
    var weights = [];

    // Store all the weights in an array
    for (var g = 0; g < pSelectedGenomes.length; g++) {
      weights.push(pSelectedGenomes[g].getWeights());
    }

    // For each genome minus the number of selected genomes divided by 2
    for (var genIndex = 0; genIndex < pGeneration.genomes.length; genIndex++) {// - (pSelectedGenomes.length / 2)
      //Pair of genome
      for (var pairIndex = 0; pairIndex < pSelectedGenomes.length; pairIndex+=2) {
        //Get the pair of genome
        var breedA;
        var breedB;
        var newWeight = [];
        //Number of genome to create per pair
        for (var i = 0; i < pGeneration.genomes.length / (pSelectedGenomes.length / 2); i++) {
          // Check which breed will be first
          var aFirst = Math.random() >= 0.5; //Random bool
          if (aFirst) {
            breedA = pSelectedGenomes[pairIndex].getWeights();
            breedB = pSelectedGenomes[pairIndex+1].getWeights();
          }
          else{
            breedA = pSelectedGenomes[pairIndex+1].getWeights();
            breedB = pSelectedGenomes[pairIndex].getWeights();
          }

          // Get a random crossover point
          var crossoverPoint = Math.floor(Math.random() * (weights[0].length + 1));

          // Create a new weight
          newWeight.push(breedA.slice(0, crossoverPoint));
          newWeight.push(breedB.slice(crossoverPoint, crossoverPoint + weights[0].length));
```

```
73            newWeight = ravel(newWeight);
74          }
75          // Add the new weight
76          children.push(newWeight);
77        }
78
79        // Assign the new weights
80        genClone.genomes[genIndex].setWeights(children[genIndex]);
81      }
82
83      // Generate a random genomes
84      /*for (var genIndex = pGeneration.genomes.length - ↩
          (pSelectedGenomes.length / 2); genIndex < ↩
          pGeneration.genomes.length; genIndex++) {
85        genClone.genomes[genIndex] = new Genome(pGeneration.topology, ↩
            pGeneration.activationFunction);
86      }*/
87
88      // Return the new generation
89      return genClone.genomes;
90    }
91  }
92
93  /**
94   * not implemented yet
95   * @extends crossover
96   */
97  class twoPointCrossover extends crossover{
98    constructor() {
99      // Parent constructor
100     super();
101   }
102
103   /**
104    * Process the crossover method
105    * @param {number} pGeneration - The generation.
106    * @param {number} pSelectedGenomes - The selected genomes.
107    * @return {genomes} The modified genomes.
108    */
109   process(pGeneration,pSelectedGenomes) {
110
111   }
112 }
113
114 /**
115  * not implemented yet
116  * @extends crossover
117  */
118 class uniformCrossover extends crossover{
119   constructor() {
120     // Parent constructor
121     super();
122   }
123
124   /**
125    * Process the crossover method
126    * @param {number} pGeneration - The generation.
127    * @param {number} pSelectedGenomes - The selected genomes.
128    * @return {genomes} The modified genomes.
129    */
130   process(pGeneration,pSelectedGenomes) {
131
132   }
133 }
```

15.17 mutation

```
1  /*************************************************************
2  * Author : De Biasi Loris
3  * Description : The mutation method class
4  * Version : 0.1
5  * Date : 16.06.2017
6  *************************************************************/
7
8  /** Abstract class for mutation method */
9  class mutation {
10   constructor() {
11     if (new.target === mutation) {
12       throw new TypeError("Cannot construct mutation instances ↩
             directly");
13     }
14   }
15
16   process(pNextGenomes) { throw new Error("Must override method"); }
17 }
18
19 /**
20  * Mutation class without rate
21  * @extends mutation
22  */
23 class mutationWithoutRate extends mutation {
24   constructor() {
25     // Parent constructor
26     super();
27   }
28
29   /**
30    * Process the mutation method
31    * @param {number} pNextGenomes - A bunch of genomes.
32    * @return {genomes} The modified genomes.
33    */
34   process(pNextGenomes) {
35     var weights = [];
36
37     // Store all the weights in an array
38     for (var g = 0; g < pNextGenomes.length; g++) {
39       weights.push(pNextGenomes[g].getWeights());
40     }
41
42     // Number of values that will change
43     var nbValuesToChange;
44
45     // Apply random mutation to each genomes
46     for (var genIndex = 0; genIndex < weights.length; genIndex++) {
47       // Get the number of values that need to change
48       var indexArray = [];
49       var alreadyExist;
50       nbValuesToChange = Math.floor(Math.random() * ↩
             (weights[genIndex].length + 1));
51
52       // While we dont have all the index that needs to change
53       while (indexArray.length < nbValuesToChange) {
54         alreadyExist = false;
```

```
55          var indexToChange = Math.floor(Math.random() * ↩
               (weights[genIndex].length + 1));
56
57          // Check if the current index has already been changed
58          for (var i = 0; i < indexArray.length; i++) {
59            if (indexToChange == indexArray[i]) {
60              alreadyExist = true;
61            }
62          }
63
64          // If the current index hasn't been changed previously
65          if (alreadyExist == false) {
66            indexArray.push(indexToChange)
67          }
68        }
69
70        // Change the weights randomly
71        for (var i = 0; i < indexArray.length; i++) {
72          weights[genIndex][indexArray[i]] += Math.random() * 0.4 - ↩
               0.2; // [-0.2...0.2]
73        }
74
75        // Apply the changes
76        pNextGenomes[genIndex].setWeights(weights[genIndex]);
77      }
78
79      return pNextGenomes;
80    }
81  }
82
83  /**
84   * Mutation class with rate
85   * @extends mutation
86   */
87  class mutationWithRate extends mutation{
88    constructor(pMutationRate) {
89      // Parent constructor
90      super();
91
92      this.rate = pMutationRate;
93    }
94
95    /**
96     * Process the mutation method
97     * @param {number} pNextGenomes - A bunch of genomes.
98     * @return {genomes} The modified genomes.
99     */
100   process(pNextGenomes) {
101     var weights = [];
102
103     // Store all the weights in an array
104     for (var g = 0; g < pNextGenomes.length; g++) {
105       weights.push(pNextGenomes[g].getWeights());
106     }
107
108     // Apply random mutation to each genomes
109     for (var genIndex = 0; genIndex < weights.length; genIndex++) {
110
111       // Change the weights randomly
112       for (var i = 0; i < weights[genIndex].length; i++) {
113         if (this.rate <= Math.random()) {
114           weights[genIndex][i] += Math.random() * 0.4 - 0.2; // ↩
                 [-0.2...0.2]
115         }
```

```
116         }
117
118         // Apply the changes
119         pNextGenomes[genIndex].setWeights(weights[genIndex]);
120       }
121
122       return pNextGenomes;
123     }
124 }
```