Rapport sur le projet de synthèse "Japet" (Février-Juin 2012)

Brice Errandonea et Manuel De Palma

13 juin 2012

Table des matières

Chapitre 1

Introduction

Le logiciel Japet est une interface graphique destinée à visualiser des "réseaux de neurones". Il nous a été commandé par Arnaud Blanchard, chercheur au département de neurocybernétique du laboratoire ETIS, à l'Université de Cergy-Pontoise.

L'ETIS (Equipes Traitement de l'Information et Systèmes) est une unité de recherche commune à l'Université de Cergy-Pontoise, au CNRS (Institut des sciences informatiques et leurs interactions) et à l'école ENSEA. Ses projets de recherches sont répartis en quatre départements :

- Architectures, Systèmes, Technologies pour les unités Reconfigurables Embarquées
- Information, Communication, Imagerie
- Indexation Multimédia et Intégration de Données
- Neurocybernétique

Le département de neurocybernétique rassemble des chercheurs et des ingénieurs qui étudient des robots et la manière dont il est possible de leur apprendre certaines choses, comme par exemple la reconnaissance de formes ou la navigation à travers une pièce comportant des obstacles. Ce travail a pour objectifs de faire avancer la science de l'intelligence artificielle, et d'en apprendre davantage sur le cerveau humain.

Les réseaux de neurones sont les principaux outils informatiques dont disposent les chercheurs pour enseigner à leurs robots. Il ne s'agit pas, bien entendu, de véritables neurones biologiques, mais de composants logiciels dont les intéractions présentent de nombreuses similitudes avec celles des neurones d'un cerveau animal.

Une expérience de neurocybernétique fait généralement intervenir plusieurs programmes informatiques, ou **scripts**, qui peuvent d'ailleurs s'exécuter sur des machines différentes. Chaque script se compose de plusieurs **groupes** de neurones, qui s'exécutent successivement et se transmettent des données à travers

des **liaisons**. Un groupe peut comporter de nombreux neurones, qui ne sont pas liés entre eux mais avec des **neurones** d'autres groupes.

Thémis et Prométhé sont les deux principaux logiciels impliqués dans ces expériences. **Thémis** lance plusieurs **Prométhés**, dont chacun va exécuter un script donné.

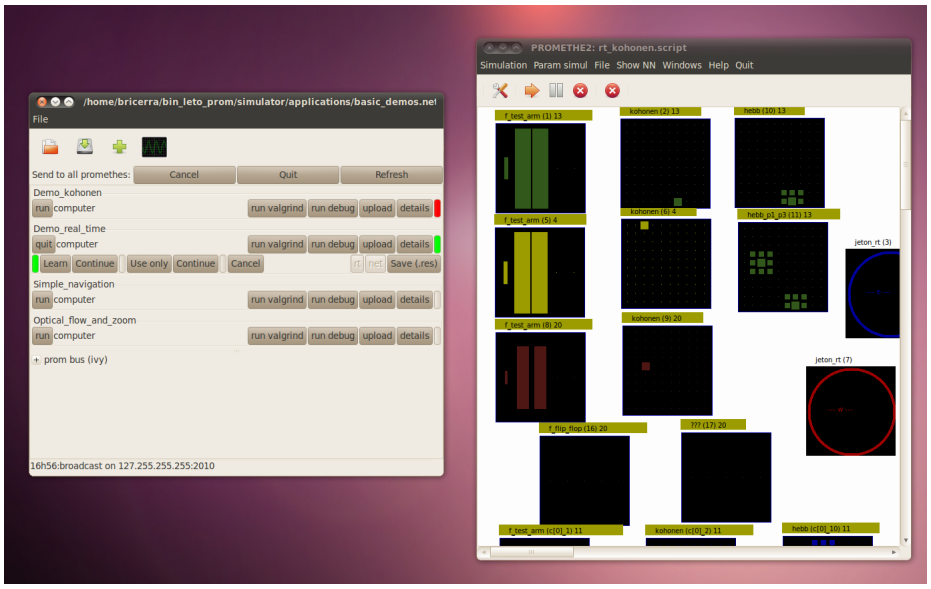


FIGURE 1.1 – Thémis (à gauche) et Prométhé (à droite)

Dans la fenêtre de droite, on observe plusieurs cadres noirs, dont chacun représente un groupe de neurones. Chaque neurone y est représenté par un rectangle dont la taille évolue au fil du temps, avec la valeur de sortie de ce neurone. Un autre logiciel important est **Coeos**, qui sert à créer les scripts en liant des groupes les uns aux autres.

Chapitre 2

Objectifs du projet Japet

Japet est une interface graphique, qui vient en complément de celles de Coeos et de Prométhés, pour que les chercheurs puissent visualiser plus facilement l'évolution des valeurs de sortie des neurones, en même temps que les liaisons entre neurones.

La première étape de notre travail a consisté à définir quelles seraient les fonctionnalités du logiciel. Nous en avons discuté de longues heures avec Arnaud Blanchard, à qui nous avons présenté successivement plusieurs maquettes.

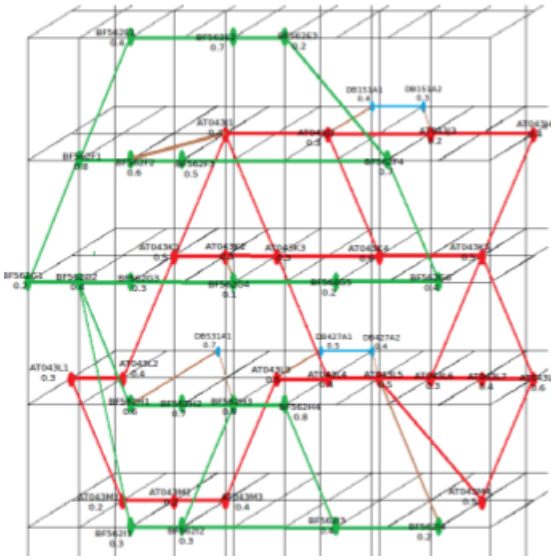


FIGURE 2.1 – Une des maquettes proposées à M. Blanchard

À l'issue de cette discussion, les fonctionnalités validées dans le cahier des charges étaient :

1. Afficher, sous la forme d'un graphe en 3D ou en perspective, les liaisons en-

tre les groupes de neurones, et indiquer quelques informations sur chaque groupe : son nom, sa fonction, sa vitesse d'apprentissage et, si possible, sa durée d'exécution.

2. Permettre à l'utilisateur de choisir quel script il veut voir dans quel plan et proposer des cases à cocher/décocher pour masquer temporairement certains scripts.

3. La coordonnée x d'un groupe dépend de l'ordre d'exécution : les premiers à s'exécuter sont à gauche. La coordonnée z dépend du script dont il fait partie. La coordonnée y est choisie librement par l'utilisateur.

4. Montrer, dans une série de petites fenêtres, l'évolution en temps réel des neurones des groupes sélectionnés par l'utilisateur. Dans chacune de ces petites fenêtres, les neurones sont représentés sous la forme d'une matrice à 2 dimensions. Un petit menu déroulant permet à l'utilisateur de choisir quelle caractéristique du neurone l'intéresse. Il peut choisir l'une des trois valeurs de sortie du neurone (s, s1 et s2), ou l'image quand le groupe en est une. Il faut qu'on puisse, à l'avenir, ajouter facilement d'autres caractéristiques à cette liste en retouchant le code. Une nuance de gris ou de couleur indique pour chaque neurone la valeur de la caractéristique demandée. L'utilisateur doit pouvoir ajuster l'échelle de ces nuances et la sauvegarder dans un fichier de préférences.

5. On veut pouvoir figer cet affichage (point 4) à tout moment. L'instant figé doit être le même pour toutes les petites fenêtres, même si les scripts concernés s'exécutent sur des machines différentes.

Ces objectifs initiaux ont évolué au cours du projet.

Nom du projet Tiré de la mythologie grecque, le nom "Japet" a été choisi pour s'harmoniser avec ceux des autres logiciels du département (en bleu) :

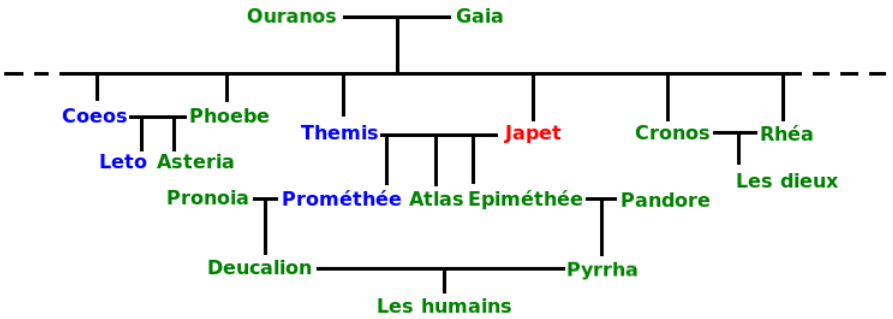


FIGURE 2.2 – Généalogie des titans

Chapitre 3

Fonctionnalités développées

Les fonctionnalités suivantes sont opérationnelles (détails pages suivantes) :
– Affichage des scripts et groupes
– Affichage des 3 types de valeurs des neurones
– Déplacement d'un script, d'un groupe ou d'une vignette de neurones
– Mise à jour continue des valeurs
– Modification des échelles et de la fréquence de rafraîchissement
– Sauvegarde et chargement des échelles

Un travail de déboguage est encore nécessaire sur :
– Figer/défiger l'affichage des neurones et revenir sur d'anciennes valeurs

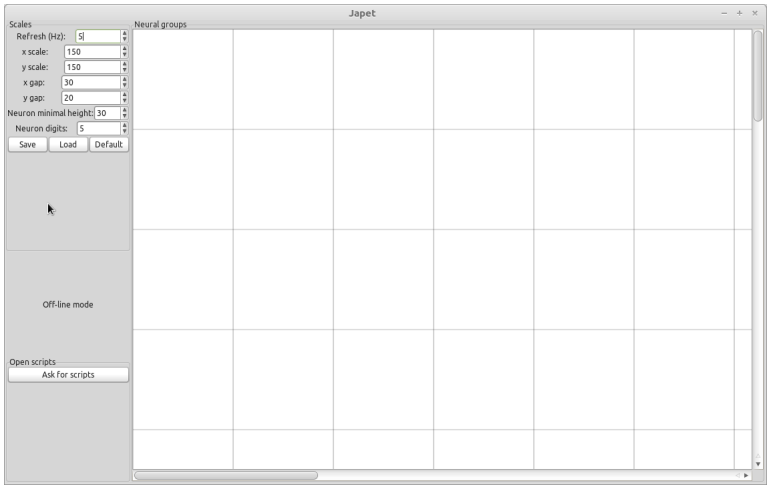


FIGURE 3.1 – Japet lors de son ouverture

La fenêtre de Japet comporte initialement 3 zones :

– Scales : les échelles
– Open scripts : les scripts ouverts (il n'y en a pas encore)

– Neural groups : c'est la zone 3D dans laquelle on verra les liaisons entre les groupes

On peut interroger tous les Prométhés connectés pour qu'ils envoient leurs scripts, puis afficher les groupes composant ces scripts dans la zone 3D.

On peut choisir dans quel plan on affiche chaque script et déplacer des groupes verticalement. On ne peut pas les déplacer horizontalement car leur abscisse est déterminée par l'ordre chronologique d'exécution.

Sur chaque groupe, on affiche son nom et sa fonction. Son taux d'apprentissage est également en mémoire et il suffirait de changer deux lignes dans le code pour l'afficher à la place de la fonction si on le voulait.

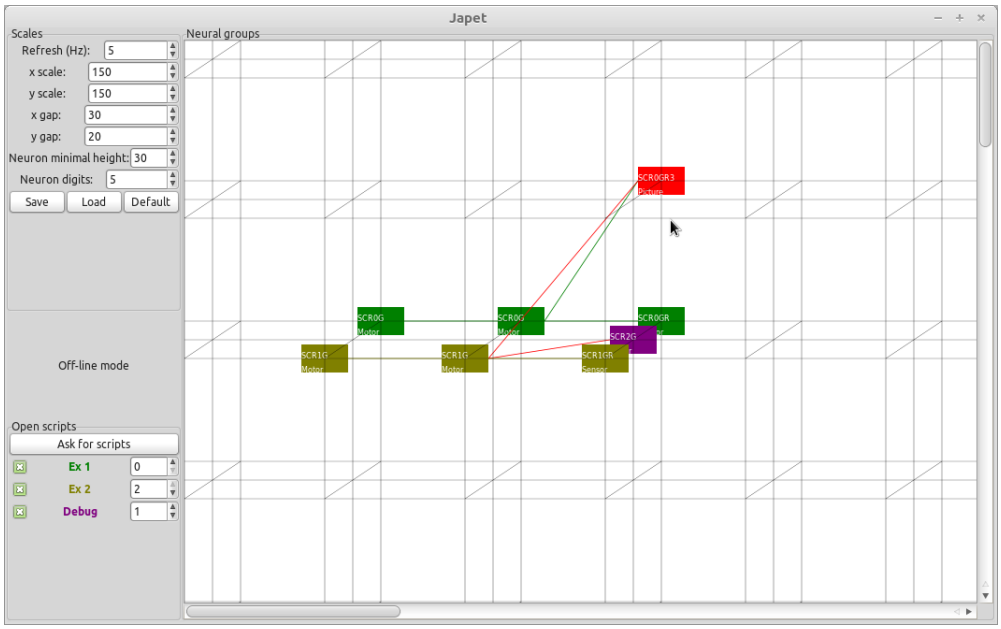


FIGURE 3.2 – Affichage des liaisons entre groupes

On peut afficher dans de petites vignettes en bas de l'écran les valeurs de sortie s, s1 et s2 des neurones d'un groupe.

Les valeurs sont écrites sur les neurones, et également représentées par des niveaux de gris.

La largeur de chaque vignette est liée au nombre de colonnes de neurones qu'elle doit afficher.

On peut sélectionner une vignette pour la fermer, pour la déplacer horizontalement, pour mettre en rouge, dans la zone 3D, le groupe correspondant à

cette vignette, et pour afficher une ligne indigo épaisse en bas de chaque vignette qui affiche les neurones du même groupe.

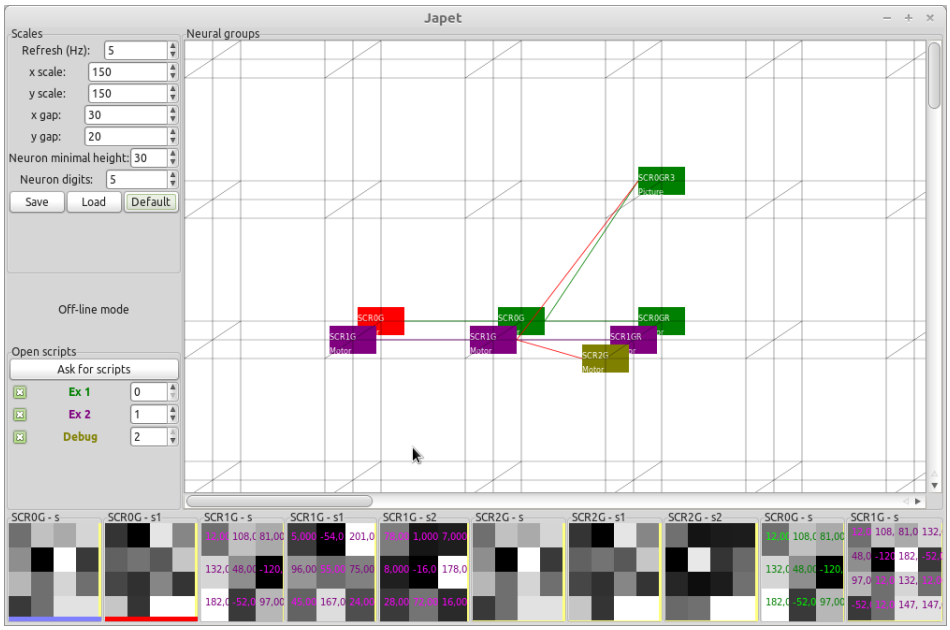


FIGURE 3.3 – Affichage des valeurs de sortie des neurones

Plusieurs échelles peuvent être modifiées et enregistrées dans un fichier de préférences, que l'on pourra recharger plus tard :

- Refresh (Hz) : Nombre de fois par seconde où l'affichage est réactualisé, pour qu'on puisse suivre l'évolution des valeurs de sortie des neurones.
- x scale, y scale, x gap, y gap : Échelles de la grille 3D. Voir image ci-dessous.
- Neuron minimal height : Hauteur minimale d'un neurone dans les vignettes.
- Neuron digits : Nombre de caractères utilisés pour afficher la valeur de sortie d'un neurone. S'il n'y a pas assez de place, la valeur n'est pas affichée. Si le nombre de caractères demandé est trop petit pour afficher la partie entère d'une valeur, des dièses sont affichés à la place de cette valeur.

Le mode d'affichage par défaut est le mode "échantillonné" (Sampled mode). On ne rafraichit pas l'écran chaque fois que Prométhé envoie des données (un oeil humain ne pourrait pas toujours suivre), mais en fonction de la fréquence de rafraichissement choisie.

Les 100 dernières valeurs de sortie de chaque neurone sont conservées en mémoire.

Nous avons développé un mode d'affichage "instantanés" (Snapshots mode) qui doit permettre de figer l'affichage et de remontrer, pas après pas, ses précédentes valeurs de sortie. Mais du travail de déboguage est encore nécessaire sur cet autre mode.

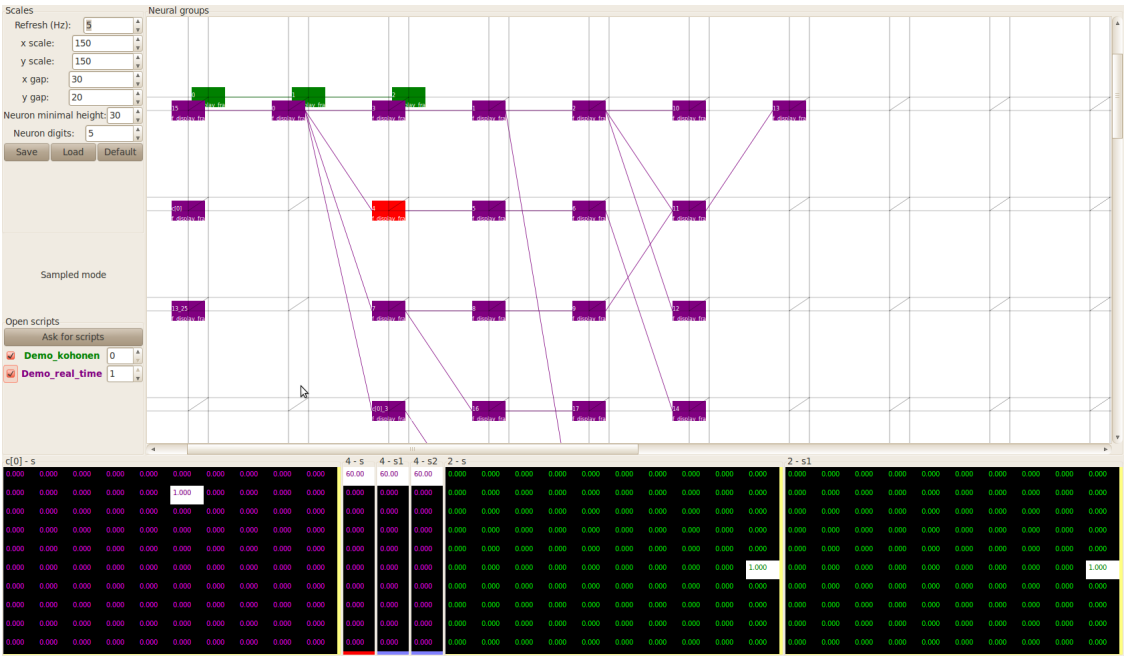


FIGURE 3.4 – Deux vrais scripts en mode "échantillonné"

Chapitre 4

Implémentation

4.1 Structures de données

On l'aura compris, les notions de script, de groupe et de neurone sont centrales dans le programme. Elles y sont représentées par des structures.

4.1.1 script

```
typedef struct script
{
    gchar* name;
    gchar* machine;
    gint z;
    gint nbGroups;
    group* groups; //Tableau des groupes du script
    gboolean displayed; //TRUE s'il faut afficher le script
} script;
```

Dans cette structure, on trouve le nom du script, l'adresse IP de la machine sur laquelle il s'exécute, la cote z du plan dans lequel on l'affiche (ou pas), le nombre de groupes dans le script, des pointeurs vers chacun de ces groupes, ainsi qu'un booléen indiquant si le script est affiché ou non.

4.1.2 group

```
typedef struct group
{
    struct script* myScript;
    gchar* name;
    gchar* function;
    gfloat learningSpeed;
    gint nbNeurons;
    gint rows;
    gint columns;
    neuron* neurons; //Tableau des neurones du groupe
    gint x;
```

```
    gint y;
    gboolean knownX; //TRUE si la coordonnée x est connue
    gboolean knownY;
    gint nbLinksTo;
    struct group** previous; //Adresses des groupes ayant des liaisons vers celui-ci
    gint sWindow[4]; //Numéro (entre 0 et NB_WINDOWS_MAX - 1) de la fenêtre où s'affichent
    gboolean justRefreshed; //TRUE si le dernier message envoyé par Prométhé a réactualisé
    gint firstNeuron; ///Numéro du premier neurone de ce groupe dans le grand tableau de t
    gint allocatedNeurons; ///Nombre de neurones déjà rangés
} group;
```

Un "struct group" comporte de nombreuses informations : un pointeur vers le script dont le groupe fait partie, un nom, une fonction, une vitesse d'apprentissage, et un nombre de neurones. Ces neurones forment dans le groupe un tableau à deux dimensions, avec un certain nombre de lignes (rows) et de colonnes (columns).

Puis, on trouve les pointeurs vers les neurones du groupe, les coordonnées x et y du groupe dans la grille tri-dimensionnelle, des booléens indiquant si ces coordonnées ont déjà été calculées, le nombre de liaisons aboutissant à ce groupe, et des pointeurs vers les groupes situés au début de ces liaisons.

Enfin, le tableau "sWindow" enregistre les numéros des vignettes qui affichent les valeurs de sortie des neurones de ce groupe. "justRefreshed" indique si ce groupe vient d'être réactualisé, "firstNeuron" est le numéro du premier neurone de ce groupe dans le tableau envoyé par Prométhé (voir la section sur les échanges réseau). "allocatedNeurons" est utilisé au moment de la création des neurones du groupe : il indique combien de neurones sont déjà rangés dans le tableau "neurons" de ce groupe.

4.1.3 neuron

```
typedef struct neuron
{
    struct group* myGroup;
    gfloat s[4]; //Valeurs du neurone (0 : s, 1 : s1, 2 : s2, 3 : pic)
    gint x;
    gint y;
    gfloat buffer[4][NB_BUFFERED_MAX];
} neuron;
```

Plus simple, la structure "neuron" comporte un pointeur vers le groupe dont ce neurone fait partie, le tableau des valeurs de sortie de ce neurone, ses coordonnées dans le groupe, ainsi qu'un grand tableau qui enregistre les précédentes valeurs de sortie (voir la section consacrée au mode "instantanés").

Chaque neurone a trois valeurs de sortie : s, s1 et s2. Nous en avons prévu une quatrième car certains groupes sont des images, dont chaque neurone est un pixel. s[3] est donc disponible pour enregistrer le niveau de gris de ce pixel.

4.1.4 Autres structures

D'autres structures, héritées de Prométhé, sont utilisées pour interpréter les informations envoyées par ce logiciel, avant des les ranger dans les structures décrites plus haut. Le fichier "reseau.h" de Prométhé, où ces structures héritées sont définies, est donc inclus dans "japet.h".

4.2 La bibliothèque graphique GTK

4.2.1 Principes généraux

Les principes généraux de GTK ressemblent beaucoup à ceux de toutes les bibliothèques graphiques. On commence par initialiser la bibliothèque avec :

```
void gtk_init(int *argc, char ***argv);
```

Puis, on crée un certain nombre de composants graphiques : des boutons, des étiquettes, des zones de dessin, etc. On emboîte ces widgets les uns dans les autres, et on y connecte des **signaux**.

Une fois tous les widgets créés, on les affiche avec la fonction

```
void gtk_widget_show_all(GtkWidget *widget);
```

où *widget, est le widget correspondant à l'ensemble de la fenêtre du logiciel.

Puis on lance une boucle infinie :

```
void gtk_main(void);
```

Sans intervention de l'utilisateur, cette boucle ne se terminera jamais. Elle attend des événements, comme par exemple de clics de souris ou l'appui sur certaines touches. Ces événements génèrent des signaux, qui appelent alors des fonctions.

La documentation officielle de GTK est disponible à l'adresse http ://developer.gnome.org/gtk/stable/. Précisons que nous utilisons la version 2 de GTK, bien que GTK 3 soit disponible, car cela nous assure que Japet tournera sans problème sur Ubuntu 10.04, le système d'exploitation de l'ETIS. La bibliothèque Cairo est également utilisée pour les dessins proprement dits.

4.2.2 L'arbre des widgets

Le schéma ci-dessous montre la manière dont les widgets de Japet s'emboitent les uns dans les autres. Le sommet de cette hiérarchie (pWindow, à gauche), est bien sûr la fenêtre générale de l'application.

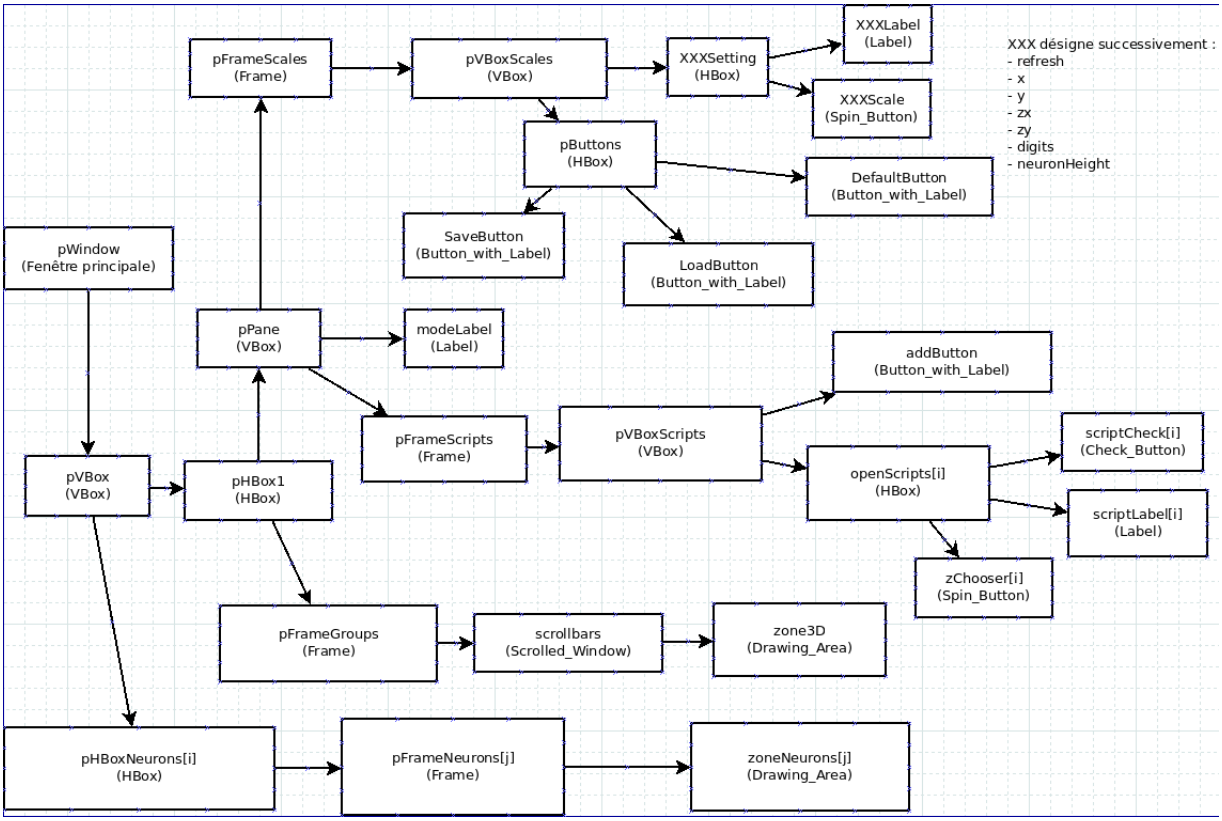


FIGURE 4.1 – Hiérarchie des widgets

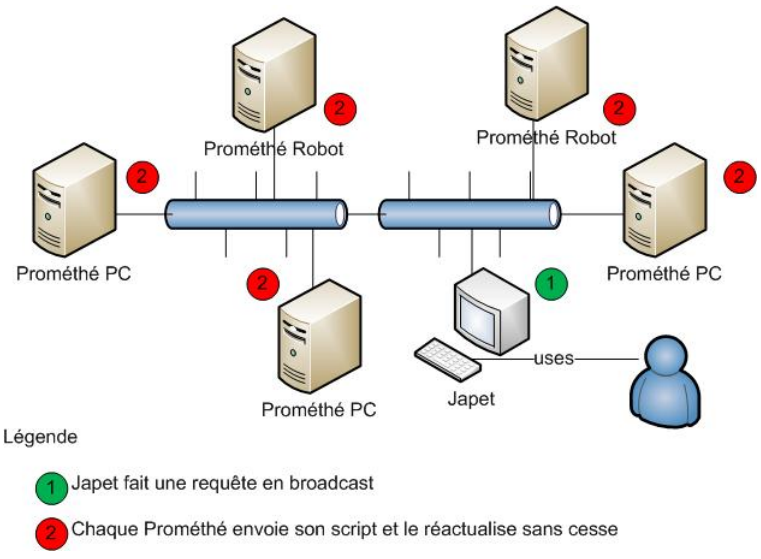
4.3 Réseau : Ivy et ENet

Pour recevoir des données d'une ou plusieurs instances de Prométhé, qui peuvent être distribuées sur plusieurs ordinateurs, Japet utilise deux protocoles réseau distincts : Ivy (plus pratique pour le broadcast) et ENet(plus rapide).

Japet commence par envoyer, via Ivy, un message en broadcast à toutes les machines du réseau. Les Prométhés qui reçoivent ce message, et qui ont été configurés pour écouter Japet, se connectent alors à lui avec Ivy, puis avec ENet.

Les échanges suivants se font avec ENet. Japet joue le rôle de serveur ENet et les Prométhés sont ses clients. Une fois connecté, chaque Prométhé envoie quatre types d'informations, sur quatre canaux différents : le nom du script qu'ils exécutent, le tableau des groupes du script, le tableau des liaisons, et celui des neurones. Japet analyse alors ces informations et les organise dans ses propres variables et tableaux. Il va pouvoir les afficher.

Chaque fois que les valeurs de sortie des neurones d'un groupe changent dans un Prométhé, ce dernier envoie le tableau des neurones de ce groupe à Japet, qui peut donc les enregistrer et mettre à jour ses propres neurones.



reseau.jpg

FIGURE 4.2 – Plusieurs Prométhés en réseau avec Japet

4.4 Un exemple d'algorithme

Voici l'un des algorithmes utilisés dans le programme. Il permet de calculer l'abscisse x de chaque groupe dans la grille tridimensionnelle, pour que les groupes qui s'exécutent avant soient toujours représentés à gauche de ceux qui s'exécutent après. Ce calcul est réalisé par la fonction récursive findX() :

```
void findX(group* g)
{
    int i, Max = 0;
    if (g->previous == NULL)
    {
      g->x = 1;
    }
    //Si ce groupe ne dépend d'aucun autre, on le met tout à gauche
    else
    {
      for (i = 0; i < g->nbLinksTo; i++)
      {
if (g->previous[i]->knownX == FALSE) findX(g->previous[i]);
if (g->previous[i]->x > Max) Max = g->previous[i]->x;
      }
      g->x = Max + 1;
    }
    g->knownX = TRUE;
}
```

Lors de la création d'un groupe, son champ "knownX" est initialisé à FALSE, car son x n'a pas encore été calculé. Une fois que tous les groupes sont créés,

et que Prométhé nous a informé sur la disposition des liaisons, on applique la fonction findX() à chaque groupe, et ce pour tous les scripts.

Les groupes qui n'ont pas de prédécesseurs sont les premiers à s'exécuter. On les place donc à gauche, avec l'abscisse x = 1. Par contre, quand un groupe à des prédecesseurs, on applique d'abord findX() à chacun d'eux, jusqu'à ce que tous aient "knownX = TRUE". On sait que le groupe *g qui nous intéresse va s'exécuter après tous ses prédécesseurs. On cherche donc lequel a la plus grande abscisse (Max) et on fixe l'abscisse de *g à (Max + 1) : un peu plus à droite que celui de ses prédécesseurs, qui est déjà le plus à droite.

Et maintenant que ce groupe a une abscisse, on met son knownX à TRUE.

4.5 Les fichiers du projet

4.5.1 Arborescence des fichiers

Les fichiers du code-source de Japet sont rangés avec ceux du code-source de Prométhé, dans un dossier intitulé "simulateur/".

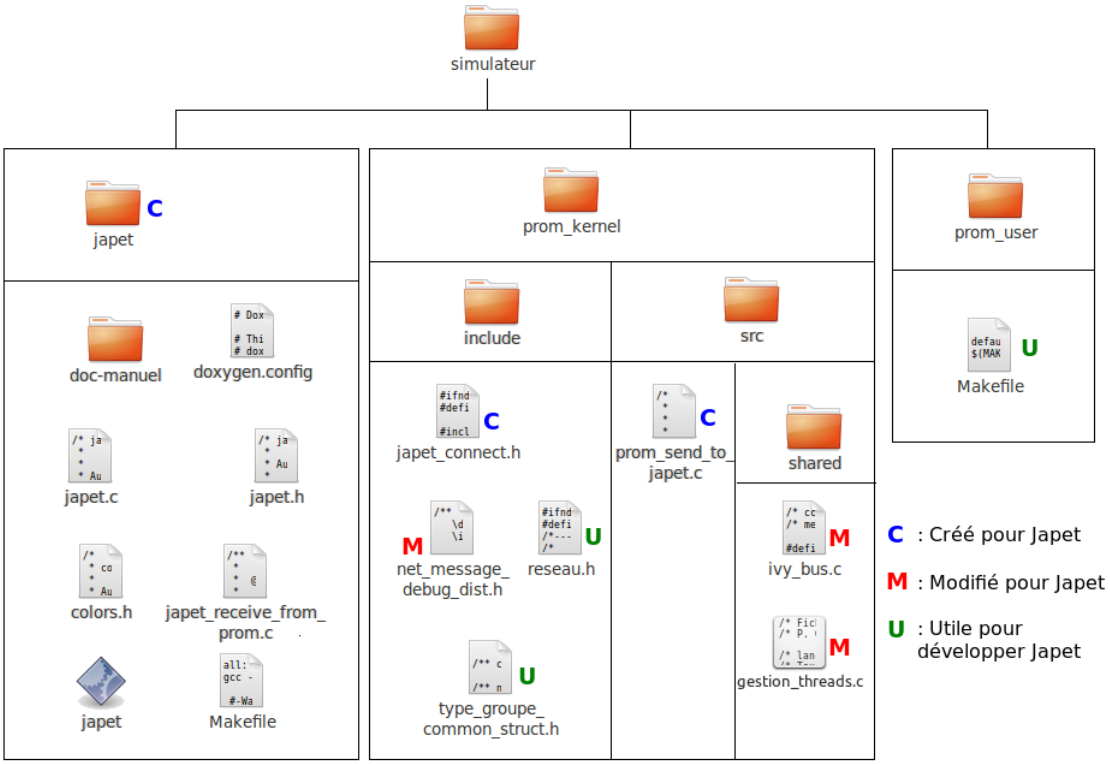


FIGURE 4.3 – Arborescence des fichiers de Japet

4.5.2 japet.c

Avec près de 1700 lignes, japet.c est un très gros fichier, qui rassemble la grande majorité du code du logiciel. Cette manière de programmer est assez inhabituelle. En général, quand un logiciel a beaucoup de code, celui-ci est éclaté entre de nombreux fichiers, eux-mêmes répartis dans plusieurs dossiers et sous-dossiers. Le problème, avec ce type d'organisation, est qu'il est souvent difficile, pour quelqu'un qui découvre le code, d'y retrouver une fonction ou une variable. Nous l'avons d'ailleurs constaté avec le code de Prométhé.

Quand tout est rassemblé dans un seul fichier, par contre, une recherche ne prend que quelques secondes et se résume généralement à un simple "Ctrl F". Mais il est alors impératif de bien structurer ce fichier. Il comporte donc plusieurs sections :

1. Variables globales : Elles sont communes à toutes les fonctions du programme. En effet, quand une de ces variables est modifiée par une fonction, il faut que ce changement soit pris en compte ailleurs. Beaucoup de ces variables globales sont des pointeurs sur des widgets (GtkWidget*), ou sur des tableaux de widgets (GtkWidget**).

2. Main : C'est bien entendu le main() du programme, qui appelle toutes les autres fonctions. C'est là que les widgets sont créés, liés les uns aux autres, et connectés à leurs signaux.

3. Initialisation : Cette petite section comporte uniquement la fonction d'initialisation, qui est appelée au tout début du main(). Sont ainsi initialisés : le tableau des scripts (scr), le tableau Index, et celui des vignettes (windowGroup), ainsi que le protocole réseau ENet. En ce qui concerne la bibliothèque GTK, elle est initialisé directement dans le main(), pour ne pas compliquer les choses.

4. Signaux : Ces fonctions sont appelées chaque fois qu'un certain événement se produit au niveau d'un certain widget. Elles prennent toutes les deux arguments suivants : (GtkWidget* pWidget, gpointer pData). pWidget est un pointeur sur le widget qui a provoqué le signal. pData est un pointeur sur une donnée que l'on passe en paramètre à la fonction. Seules quelques fonctions utilisent réellement ce pData.

5. "Constructeurs" : Japet n'est pas un programme orienté objet. Mais nous repris ce concept de fonctions appelées chaque fois qu'on veut créer une structure script, group ou neuron, ou encore lorsqu'on veut les mettre à jour ou les détruire. On trouvera donc dans cette rubrique les fonctions newScript(), destroyScript(), destroyAllScripts(), newGroup(), newNeuron() et updateNeuron().

6. Autres fonctions : On trouve ici la fonction findX() détaillée plus haut, son homologue findY(), et diverses fonctions consacrées à l'affichage ou à la sauvegarde des échelles.

4.5.3 japet.h

japet.h est le seul fichier inclus dans japet.c. Lui, en revanche, inclut des nombreuses bibliothèques : GTK, Cairo, ENet, semaphore.h, time.h, et des fichiers comme reseau.h, qui viennent du code de Prométhé. C'est là que sont déclarées les structures de données script, group et neuron. On y définit également un certain nombre de constantes, et les prototypes de fonctions de japet.c. Les variables globales de japet.c sont déclarées ici comme des variables externes. Cela permet de les utiliser dans d'autres fichiers.

4.5.4 Autres fichiers

```
simulateur/prom_kernel/include/reseau.h
simulateur/prom_kernel/include/type_groupe_common_struct.h

simulateur/prom_kernel/src/prom_send_to_japet.c
simulateur/japet/japet_receive_from_prom.c
simulateur/prom_kernel/include/japet_connect.h
simulateur/prom_kernel/include/net_message_debug_dist.h
simulateur/prom_kernel/src/shared/ivy_bus.c
simulateur/prom_kernel/src/shared/gestion_thread.c
```

Les deux premiers fichiers ci-dessus contiennent les déclarations des structures de Prométhé. Japet en utilise certaines pour analyser ce que Prométhé lui envoie.

Les autres servent justement à transférer des données de Prométhé vers Japet. Leurs noms sont volontairement assez évocateurs.

Chapitre 5

Poursuite du projet

Japet constitue un point de départ, déjà fonctionnel, pour l'élaboration d'un nouvel outil de visualisation, complémentaire avec ceux de Prométhé et de Coeos. M. Blanchard semble le juger suffisamment prometteur pour avoir envie de le poursuivre. En effet, il a d'ores et déjà confié à un nouveau stagiaire (étudiant en licence), la tâche de poursuivre le développement des fonctionnalités d'affichage de Japet.

Il nous revient donc de faciliter la continuation du projet, en expliquant à ce stagiaire ce que nous avons réalisé, en proposant des améliorations, et en fournissant une documentation, qui servira également à tous ceux qui, après lui, travailleront sur Japet.

5.1 Améliorations possibles

- Afficher les liaisons entre groupes appartenant à des scripts différents. Ces liaisons ne figurent pas dans le paquet actuellement envoyé par Prométhé, qui les gère différemment des liaisons intra-script.

- Afficher les liaisons secondaires, dont le groupe destinataire peut être situé chronologiquement avant le groupe source. Là, je ne sais pas comment faire pour que ce soit clair et joli. Peut-être de grandes flèches courbes, en exploitant la troisième dimension.

- Ajouter une deuxième rangée de vignettes, et éventuellement une troisième. On pourra alors déplacer les vignettes d'une rangée à l'autre. Les touches "Flèche haut" et "Flèche bas" ont été réservées dans ce but.

- Afficher les liaisons entre neurones : quand on clique sur un neurone, afficher en nuances de rouge (au lieu des nuances de gris) les neurones d'autres groupes qui lui sont liés. Bien entendu, on ne les verra que s'ils sont, eux aussi, affichés dans une autre vignette.

- En plus du nom et de la fonction du groupe, ou à la place de cette dernière, afficher la vitesse d'apprentissage, qui est déjà disponible dans la structure

`group`. M. Blanchard ne semble cependant pas y tenir tant que ça.

5.2 Documentation

Le présent rapport, avec son annexe, constitue la première pierre de la documentation du projet. C'est sans doute lui qu'il faut commencer par lire pour s'initier à Japet.

Nous avons également produit une documentation de type **doxygen**, qui explique chacune des fonctions du programme.

Enfin, le code source est lui-même abondamment commenté.

Chapitre 6

Conclusion

Japet est un gros projet. Toutes les estimations que nous avions faites en cours de gestion de projet indiquaient qu'il ne serait pas possible de le terminer avant le mois d'août. Malgré cela, nous avons atteint la plupart des objectifs du cahier des charges et mis au point un logiciel qui, une fois terminé, devrait être très utile aux chercheurs du département de neurocybernétique de l'ETIS, pour visualiser et déboguer leurs réseaux de neurones.

Ce défi nous a poussé à exploiter à fond nos compétences en programmation, dans les domaines de l'interface graphique, de l'algorithmique et du réseau.

Nous sommes heureux d'avoir ainsi apporté notre pierre à la Recherche, et nous souhaitons bonne chance à Ludovic pour prendre le relais, ainsi qu'à ses futurs successeurs, que nous espérons nombreux.

Chapitre 7

Annexe : Manuel d'utilisation

7.1 Lancement

Le logiciel se présente sous la forme d'un fichier exécutable appelé `japet`. Il est destiné au système d'exploitation utilisé à l'ETIS : Linux. Son code-source est accompagné d'un fichier `Makefile`, qui permet, en cas de besoin, de le re-compiler.

Pour le lancer depuis la console, on va dans le dossier contenant l'exécutable et on tape `./japet`. Pour ailleurs, pour pouvoir travailler avec Japet, Thémis et Prométhé doivent être lancés avec l'option `-i japet`.

On voit alors ceci :

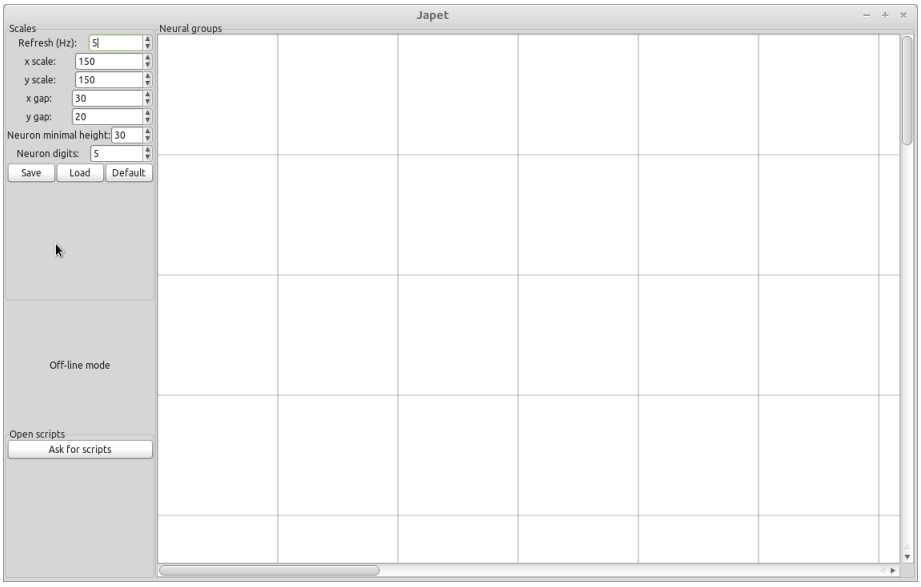


FIGURE 7.1 – Japet lors de son ouverture

Pour l'instant, la fenêtre comporte donc 3 zones :

- Scales : les échelles
- Open scripts : les scripts ouverts (il n'y en a pas encore)
- Neural groups : c'est la zone 3D dans laquelle on verra les liaisons entre les groupes

Il faut commencer par cliquer sur le bouton "Ask for scripts". Japet envoie alors un message à tous les Prométhés connectés et chacun envoie son script.

Attention : Si, plus tard dans l'exécution, on clique à nouveau sur ce bouton, c'est comme si on relançait Japet. Les scripts précédents sont effacés de la mémoire de Japet, avec tous leurs instantanés, et on interroge à nouveau tous les Prométhés. Seules les échelles sont conservées.

7.2 Scripts, groupes et plans

Quand on clique sur "Ask for scripts", les scripts disponibles apparaissent dans la zone "Open scripts" :

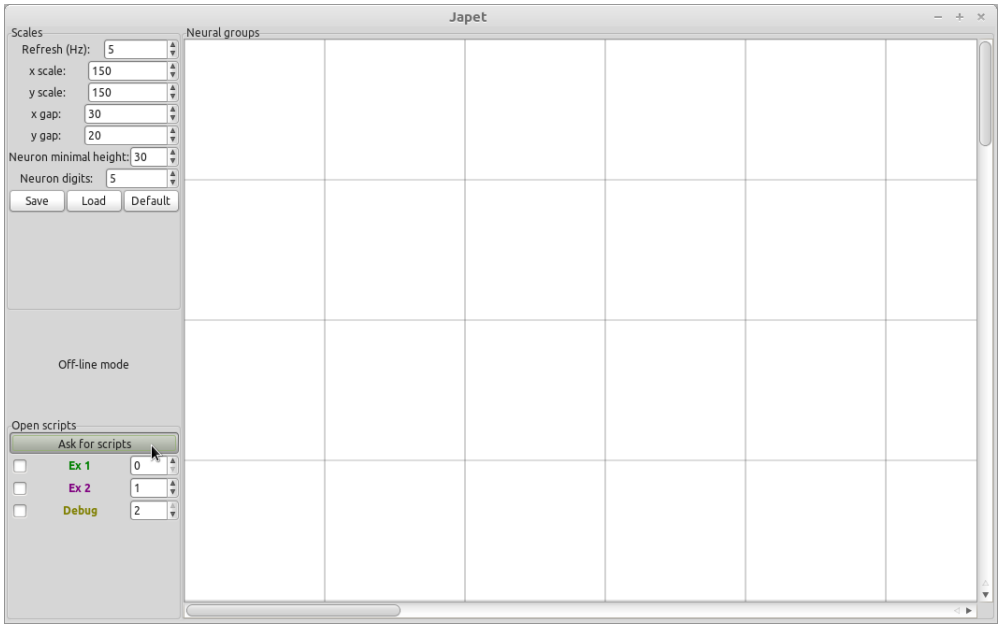


FIGURE 7.2 – Réception des scripts

Dans cet exemple, trois instances de Prométhé sont connectées. La première exécute un script appelé `Ex 1` (première exemple), la deuxième un script `Ex 2`, et la troisième exécute un petit script de déboguage : `Debug`.

On va maintenant cocher les scripts que l'on souhaite afficher.

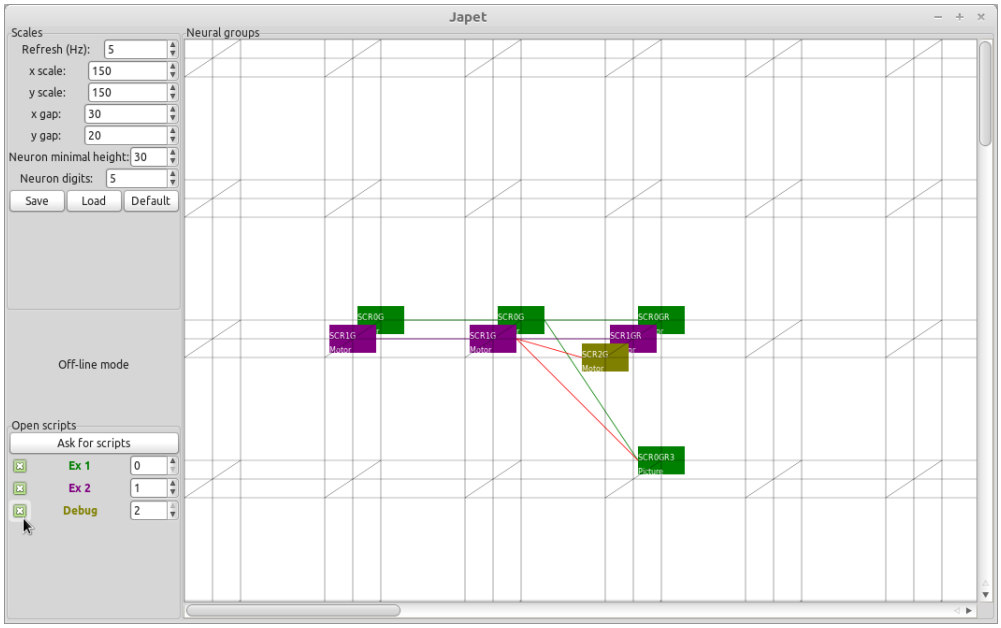


FIGURE 7.3 – Affichage des scripts

Les scripts cochés s'affichent alors dans la zone « Neural groups », dans un repère tridimensionnel. Chaque rectangle représente un groupe de neurones. Il affiche le nom du groupe et sa fonction.

Par défaut, chaque script est affiché dans un plan différent. Ici, il y a trois scripts, donc trois plans. Mais lorsqu'il y a plusieurs petits scripts de déboguage, avec seulement 1 ou 2 groupes dans chacun, on peut avoir intérêt à les regrouper dans un même plan. Dans le panneau Open scripts, on peut donc choisir, en cliquant sur les petites flèches, le numéro du plan dans lequel on veut afficher chaque script.

Une couleur, toujours assez sombre (on verra pourquoi), est associée à chaque numéro de plan :

- 0 : vert foncé
- 1 : violet prune
- 2 : jaune-marron
- 3 : bleu marine
- 4 : bleu paon (cyan foncé)
- 5 : orange-marron
- 6 : gris foncé

Il ne peut pas y avoir plus de 7 plans (on ne verrait rien). Si on veut afficher plus de scripts, il faut en mettre plusieurs dans un plan.

Par exemple, mettons Ex 2 dans le plan 2 et Debug dans le plan 1 :

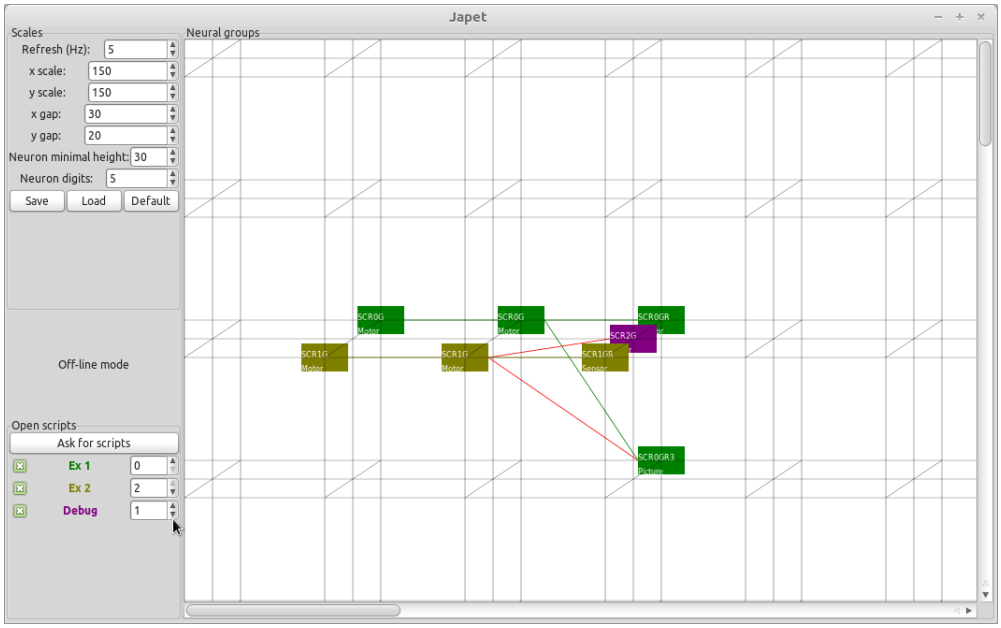


FIGURE 7.4 – Changement de plan

Par contre, on ne peut pas aller dans le plan 3 (le 4ème, donc) s'il n'y a que 3 scripts au total.

On remarquera que la couleur dans laquelle le nom d'un script est écrit à gauche s'adapte à la couleur de son plan.

Les liaisons entre deux groupes d'un même script sont de la même couleur que ce script. En revanche, les liaisons entre deux groupes appartenant à des scripts différents sont systématiquement rouges, même si les deux scripts sont affichés dans le même plan.

Dans la zone 3D, chaque groupe a 3 coordonnées : x, y et z.

- L'abscisse x est calculée par Japet pour que les groupes soient affichés dans l'ordre chronologique : ceux qui s'exécutent en premier sont à gauche et ceux qui s'exécutent plus tard sont à droite. Il n'est donc pas nécessaire de mettre des flèches sur les liaisons : elles vont toujours de gauche à droite.

- La cote z est le numéro du plan. On a vu comment la changer.

- L'ordonnée y n'a pas de signification particulière et peut être changée librement par l'utilisateur.

Pour modifier l'ordonnée y d'un groupe, il faut d'abord le sélectionner en cliquant dessus avec le bouton gauche. Le groupe devient alors rouge. Puis, on le déplace verticalement avec les touches "Page Up" et "Page Down".

Par exemple, on clique sur le groupe SCR0GR3 (en bas à droite). Cela le sélectionne, donc il devient rouge. Puis on appuie **une seule fois** sur la touche "Page Up".

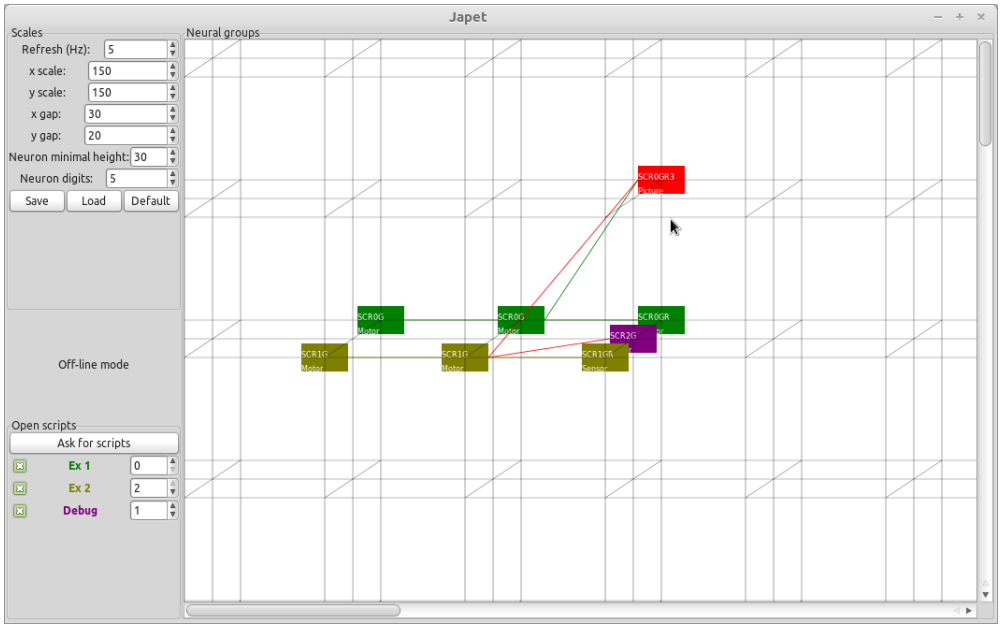


FIGURE 7.5 – Sélection et déplacement d'un groupe

On a demandé au groupe SCR0GR3 de monter. Mais comme la case au dessus de lui était déjà occupée, il est monté de deux cases. Les liaisons concernant le groupe déplacé suivent.

Il n'y a qu'un seul groupe sélectionné à la fois. Et si on clique dans le blanc, en dehors des groupes, il n'y en a plus aucun.

7.3 Neurones et vignettes

On veut maintenant connaître les valeurs de sortie des neurones d'un groupe, par exemple le premier groupe du script Ex 1 (en haut à gauche). Pour cela, on fait un clic droit dans ce groupe.

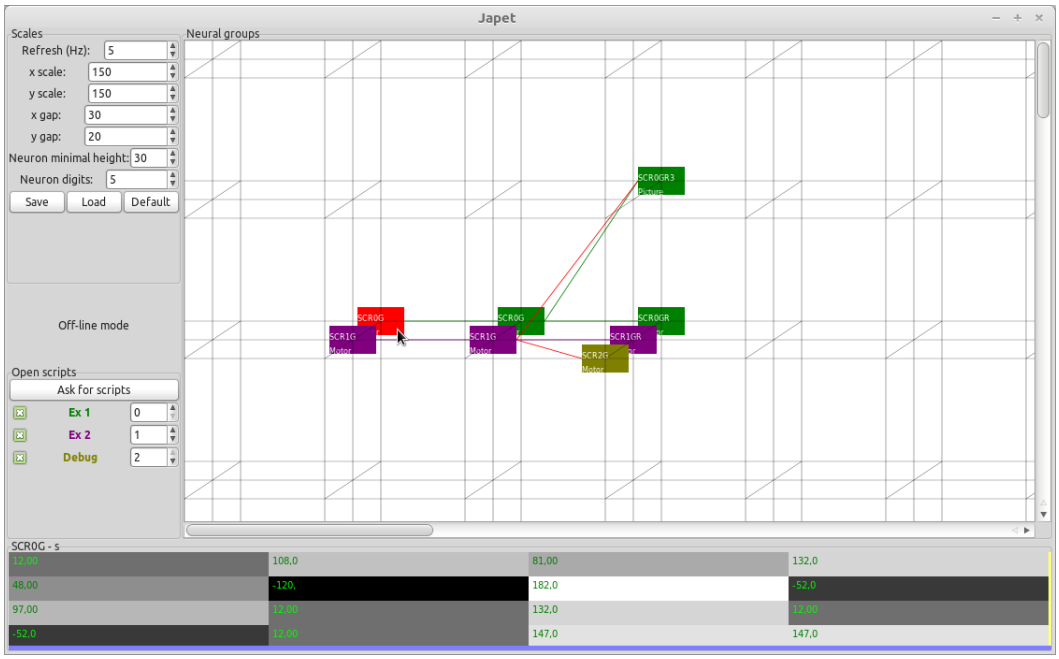


FIGURE 7.6 – Affichage des neurones d'un groupe

Non, seulement, le groupe est sélectionné, mais on voit surtout qu'une vignette s'ouvre en bas de l'écran. Elle montre les valeurs de sortie s des neurones de ce groupe.

Ce groupe comporte 16 neurones, répartis en 4 lignes et 4 colonnes. Chacun est représenté par un rectangle. Celui qui a la plus grande valeur s apparaît en blanc. Celui qui a la plus petite apparaît en noir. Les autres sont en niveaux de gris.

De plus, la valeur de s est écrite sur chaque neurone, dans la couleur du groupe, ou dans une version plus claire de la même couleur si le neurone est sombre (pour faciliter la lecture).

Pour mieux comprendre la signification de la ligne indigo en dessous, faisons tout de suite un clic droit sur un autre groupe (au milieu).

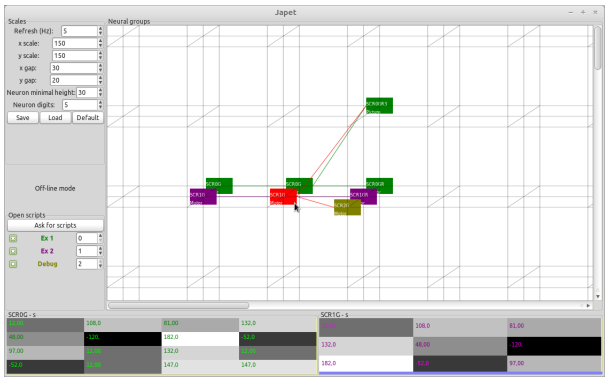


FIGURE 7.7 – Ouverture d'une deuxième vignette

La zone du bas est maintenant partagée entre deux vignettes : une pour chaque groupe. Le nouveau comporte 9 neurones : 3 lignes et 3 colonnes.

La répartition n'est pas égalitaire : la vignette de gauche a 4 colonnes de neurones à afficher, donc elle occupe plus de place que celle de droite, qui n'en a que 3.

La ligne indigo en bas signifie que cette vignette affiche les neurones du groupe qui est actuellement sélectionné dans la fenêtre principale.

Si on veut voir les autres valeurs de sortie de ces neurones, on peut faire d'autres clics droits sur le groupe. En effet, chaque neurone a trois valeurs de sorties différentes : s, s1 et s2.

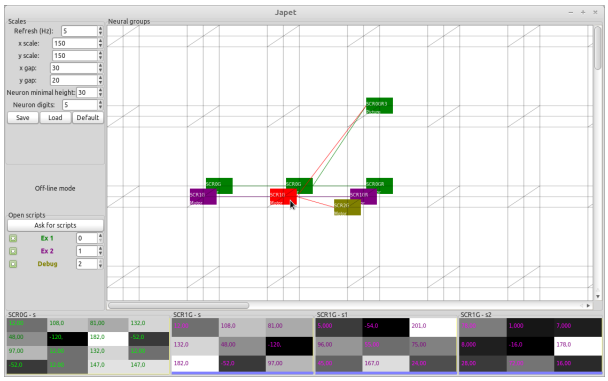


FIGURE 7.8 – s, s1 et s2

Au deuxième clic droit sur un groupe, une vignette s'ouvre pour montrer les valeurs de sortie s1 de ses neurones. Au troisième, une nouvelle vignette montre les valeurs de sortie s2 de ses neurones.

Par contre, si on continue à faire des clics droits sur ce groupe alors que ses

trois vignettes sont déjà affichées, il ne se passe plus rien.

Les 3 vignettes de droite ont toutes une ligne indigo en bas car elles concernent toutes le groupe sélectionné. Celle de gauche n'en a pas car elle concerne un autre groupe.

Continuons à afficher des petites fenêtres, en faisant des clics droits sur plusieurs groupes :

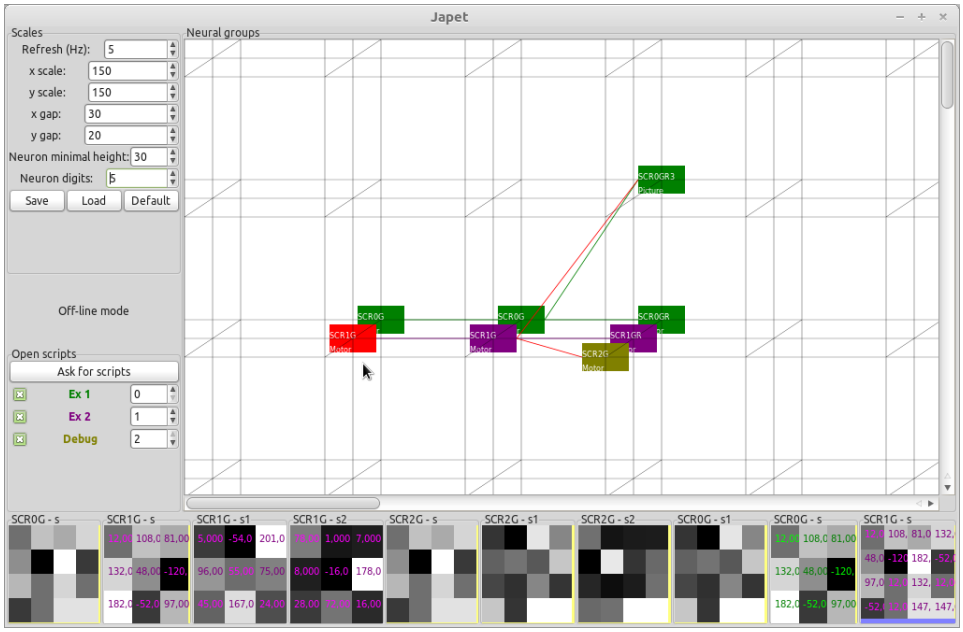


FIGURE 7.9 – Beaucoup de vignettes

Maintenant qu'il y a beaucoup de vignettes, certaines n'ont plus la place d'afficher les valeurs des neurones. Donc elles ne les affichent plus. Mais les niveaux de gris continuent à nous montrer quels neurones ont les plus grandes ou les plus petites valeurs.

Ceci dit, on n'est peut-être pas obligés d'afficher autant de chiffres après la virgule à chaque fois. C'est le moment de nous intéresser aux échelles. Nous reviendrons ensuite sur les petites fenêtres.

7.4 Les échelles

Nous n'avons pas encore parlé de la zone « Scales » en haut à gauche. Elle comporte 6 valeurs d'échelles, dont chacune peut être ajustée en cliquant sur les petites flèches à sa droite.

La dernière échelle « Neuron digits » détermine combien de caractères on utilise pour afficher les valeurs s, s1 et s2, en comptant la virgule et l'éventuel signe -. Sur la capture d'écran précédente, « Neuron digits » valait 5. Réduisons-le à 3 :

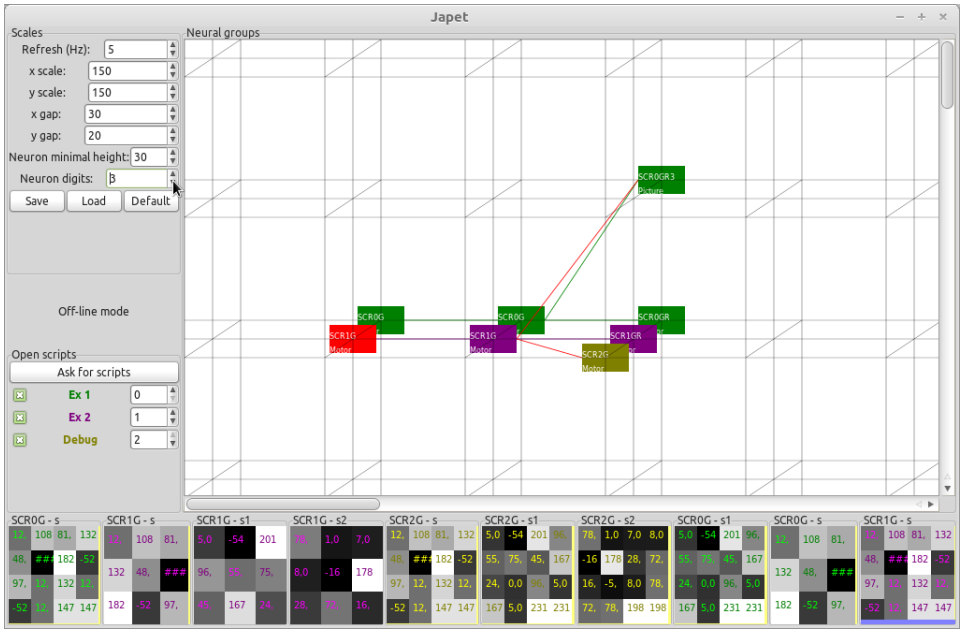


FIGURE 7.10 – 3 caractères par valeur de sortie

Ainsi, on a de nouveau assez de place pour afficher toutes les valeurs.

Mais cette limitation à 3 caractères ne présente pas que des avantages. Par exemple, prenons le neurone noir de la deuxième petite fenêtre à partir de la gauche. Sa valeur est -120. Sur la capture précédente, avec 5 caractères, on lisait « -120, ». Mais, avec seulement 3 caractères, il est impossible d'afficher entièrement sa partie entière. On obtiendrait « -12 », ce qui serait faux. Donc, on affiche des dièses à la place.

L'échelle « Neuron minimal height » impose une hauteur minimale pour chaque neurone. La hauteur des petites fenêtres en bas est déterminée par cette contrainte. Ici, il n'y a pas beaucoup de lignes de neurones à afficher donc il n'y a pas de problème. Mais si on en avait 50 ou 100, l'écran ne serait peut-être pas assez grand. Il est donc essentiel de pouvoir réduire la hauteur minimale de chaque neurone.

Bien entendu, si on la réduit beaucoup, il peut redevenir impossible d'afficher les valeurs des neurones.

Les échelles x scale, y scale, x gap et y gap concernent la zone 3D :

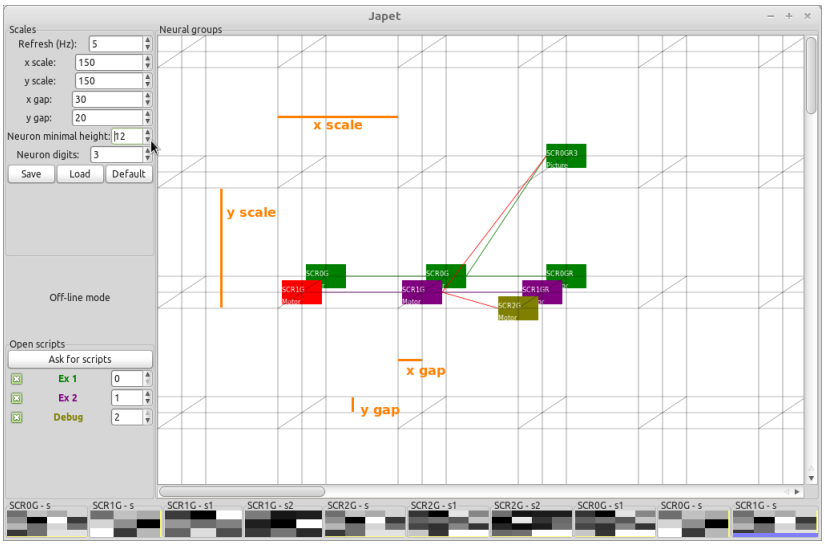


FIGURE 7.11 – Les échelles

En les modifiant, on peut obtenir, par exemple :

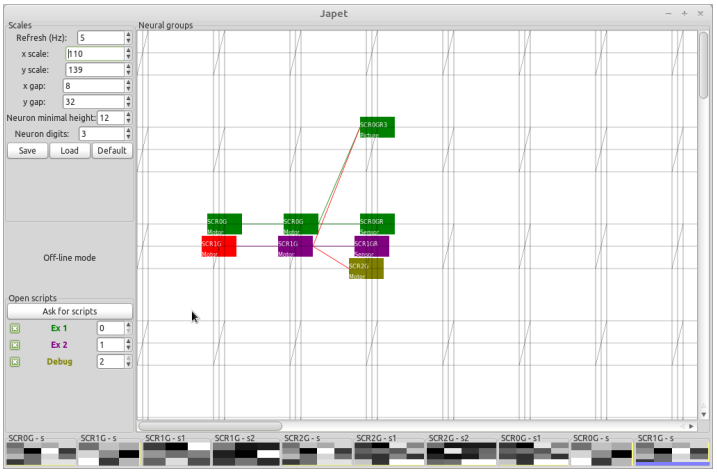


FIGURE 7.12 – Changement d'échelles

Quand on a trouvé les échelles dont on rêvait, on peut les enregistrer dans un fichier en cliquant sur le bouton "Save".
Plus tard, vous pourrez recharger ce fichier en cliquant sur le bouton "Load"

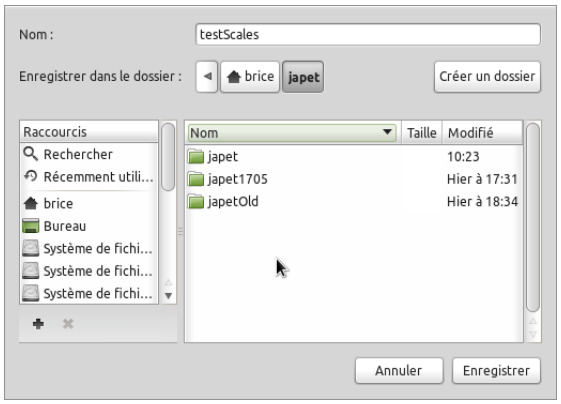


FIGURE 7.13 – Sauvegarde des échelles

et retrouver ces échelles.

Pour revenir aux échelles par défaut, il suffit de cliquer sur le bouton « Default ». Faisons-le maintenant. On retrouve l'écran correspondant à la dernière capture du chapitre précédent.

7.5 Manipuler les vignettes

Vous ne savez plus à quel groupe correspond la troisième vignette en partant de la droite ? Vous pouvez lire son titre, bien entendu. Mais vous pouvez aussi la sélectionner en faisant un clic gauche dedans.

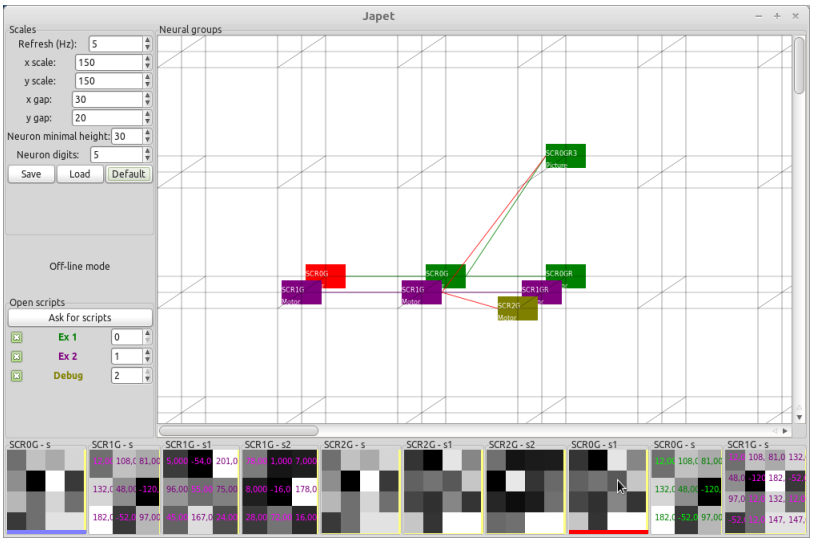


FIGURE 7.14 – Sélection d'une vignette

Une ligne rouge surgit en bas de cette vignette, car elle est maintenant sélectionnée. Le groupe correspondant est également sélectionné, donc on le repère tout de suite dans la zone 3D.

Enfin, on voit une ligne indigo en bas de la première vignette à gauche, car elle aussi concerne ce groupe.

Compte-tenu du point commun entre ces deux vignettes, on peut vouloir les regrouper. On utilise les flèches « Gauche » et « Droite » du clavier pour déplacer la vignette sélectionnée.

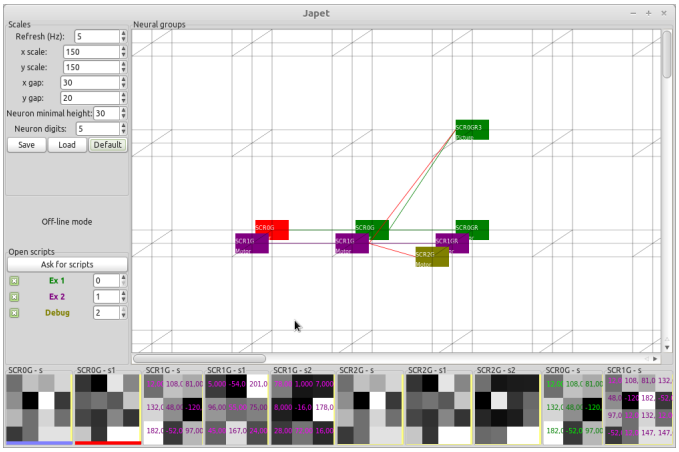


FIGURE 7.15 – Déplacer une vignette

Et maintenant, faisons un peu le ménage. Imaginons, par exemple, qu'on ne s'intéresse qu'aux valeurs s1. On peut fermer les autres petites fenêtres en faisant un clic droit dans chacune :

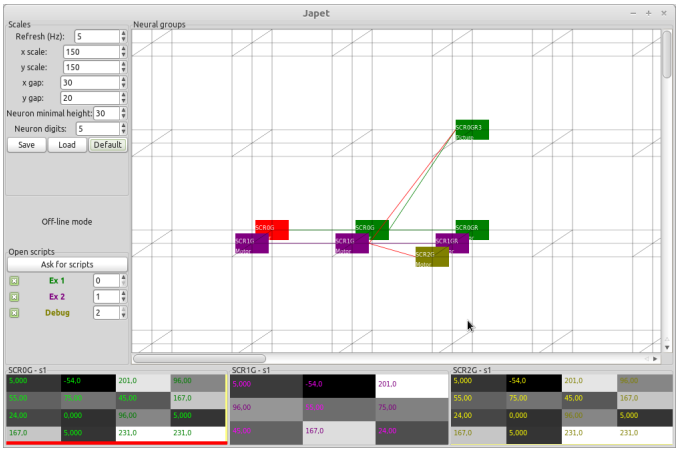


FIGURE 7.16 – Fermer des vignettes

Si on change maintenant les valeurs des z dans le panneau "Open scripts", tout s'adapte bien :

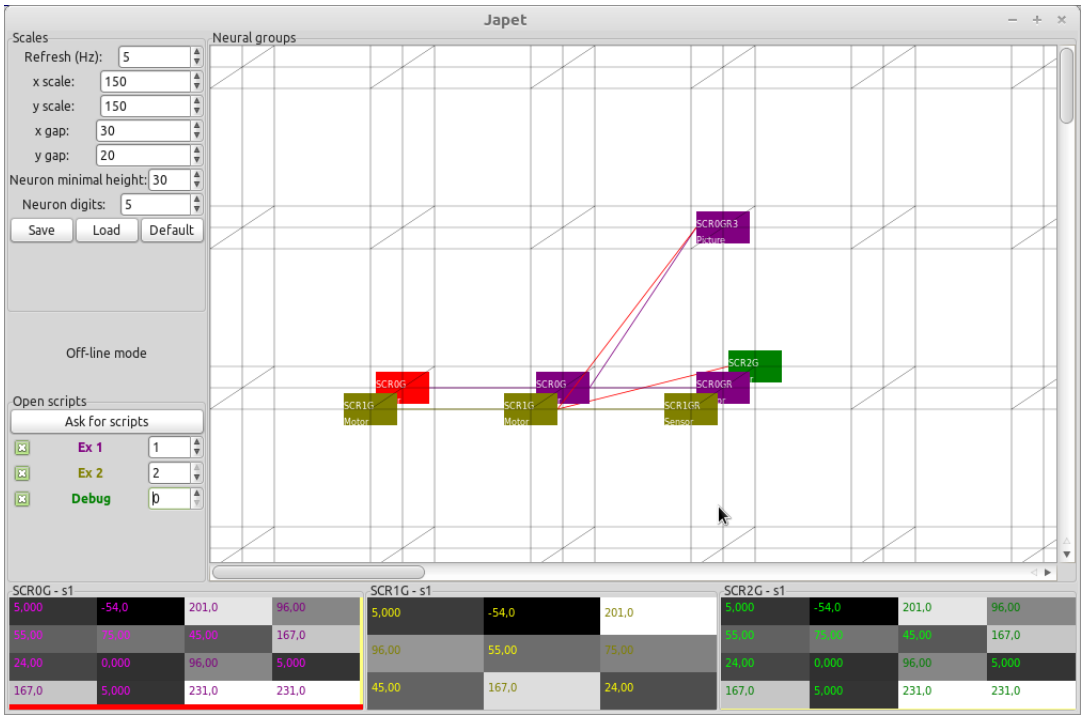


FIGURE 7.17 – Changement de plan

Et si on clique sur "Ask for scripts", les vignettes disparaissent, et tout repart à zéro.

7.6 Le mode "instantanés" (Snapshots mode)

Il n'est pas encore fonctionnel. L'idée est de figer l'affichage lorqu'on appuie sur la barre d'espace. On passe alors en mode "Snapshots". À gauche de l'écran, on peut lire "1/100", ce qui signifie que, sur 100 instantanés enregistrés en mémoire, on regarde actuellement le premier. Les touches + et - devraient permettre de passer d'un instantané à l'autre. Sur chacun, les groupes dont les neurones viennent d'être réactualisés sont affichés dans une couleur plus claire. C'est d'ailleurs pour cette raison que toutes les couleurs habituelles sont relativement sombres.