université de BORDEAUX

Projet de Recherche et Développement :

Élaboration d'un modèle computationnel modélisant le cycle éveil/sommeil chez le rongeur

---

Rapport

Client :

Héricé Charlotte, PhD
Attaché Temporaire d'Enseignement et de Recherche (ATER)
Institut Chimie et Biologie des Membranes et des Nano-objets (CBMN)
Université de Bordeaux

Auteurs :

Bielle Paul
Denet Lola
Guinot Charles
Hui Tongyuxuan
Niu Wenli
Étudiants en Master 1 de Bio-informatique
Université de Bordeaux

29 Mai 2020

Table des matières

Remerciements

Nous tenons à remercier le Dr. Charlotte Héricé de l'opportunité qu'elle nous a offerte en nous proposant et en supervisant ce projet. Nous lui exprimons toute notre gratitude pour la patience et la disponibilité dont elle a fait preuve tout au long de ce travail mais aussi pour nous avoir partagé son expertise.

Nous remercions également le Dr. Marie Beurton-Aimar qui a dirigé cet enseignement, nous permettant ainsi de faire un pas de plus vers la profesionnalisation. Nous lui sommes également reconnaissants de son écoute et de sa bienveillance en cette période sanitaire inédite et difficile.

Introduction

Il est établi que l'être humain passe un tiers de sa vie à dormir. Cela implique que le sommeil est un aspect majeur de la vie et de la santé. Cependant, la communauté scientifique ne connaît pas encore tous les mécanismes impliqués dans le système de régulation du sommeil. Les études et expérimentations menées permettent des avancées dans la compréhension de ce phénomène. Ce travail consiste à s'intéresser à la compréhension des mécanismes neurobiologiques et le rôle des régions cérébrales impliquées dans la régulation des cycles du sommeil. Pour cela, les états du sommeil et d'éveil, ainsi que leurs interactions avec les différentes populations neuronales impliquées, vont être simulés en développant un réseau de neurones artificiels qui permettrait d'approfondir l'étude de la régulation du sommeil.

Le modèle computationnel à réaliser sera basé sur des études publiées et validées par la communauté scientifique. Il permettra donc l'étude des états du sommeil chez le rongeur et leur processus de régulation. En effet, ces mammifères présentent une alternance de cycles plus fréquente que l'Homme ce qui permet d'optimiser le temps des expériences.

De manière générale, un tel modèle permet de simuler, avec une certaine abstraction, le comportement neuronal des rongeurs (pour notre sujet). Cela permet de tester un grand nombre d'hypothèses avant l'expérimentation en laboratoire, mais également de réaliser des prédictions pouvant guider l'expérimentaliste dans ses démarches.

Ce projet de recherche et de développement porte sur l'étude et l'élaboration d'un modèle computationnel qui sera accessible en ligne sur GitHub [1]. Il est encadré par le Dr Héricé Charlotte dont le travail est concentré sur la dynamique neuronale en lien avec le cycle de sommeil/éveil chez le rongeur. Ce travail aura pour but de reproduire ces cycles, sur une échelle de temps donnée, grâce à un modèle à trois populations et à six populations de neurones, qui se rapprochera un peu plus de la réalité biologique. Pour cela, les études publiées dans le domaine d'intérêt ainsi que les programmes à la base de ce travail seront abordés. Les objectifs et méthodes de conception de ce projet ainsi que les besoins seront repris brièvement. Enfin, les étapes du développement et les améliorations potentielles du modèle computationnel seront approfondies.

1. `https://github.com/lola-denet/PdP_Sommeil_Rongeur`

Chapitre 1

Analyses et Conception

1.1 État de l'art

L'état de sommeil se divise généralement en deux stades bien distincts : le sommeil paradoxal, également connu sous nom de sommeil REM (*Rapid Eye Movement*) et le sommeil NREM (*Non-Rapid Eye Movement sleep*) également appelé sommeil profond. Les différentes EEG (électroencéphalographie) réalisées ont alors montré des similarités entre l'état d'éveil et celui du sommeil paradoxal (REM), traduites par la présence d'une activité de faible amplitude à fréquence élevée. En revanche, le sommeil NREM se caractérise plutôt par des oscillations de plus basse fréquence à plus haute amplitude [Rasch et Born, 2013]. Le sommeil NREM serait alors subdivisé en plusieurs phases de sommeil caractérisées par différentes classes d'oscillations [Iber *et al.*, 2007]. Ce dernier aspect ne sera pas développé dans la suite du projet.

D'autre part, les multiples transitions d'états peuvent être étroitement liées à différents niveaux de neurotransmetteurs dans ces zones, jouant alors un rôle important tant dans l'initiation que dans le maintien du comportement éveil-sommeil [Lydic et Baghdoyan, 1993]. Par exemple, les recherches sur l'un des principaux neurotransmetteurs du système nerveux central chez les mammifères, (GABA=acide γ-aminobutyrique), ont permis de mettre en évidence l'implication des populations GABAergiques dans la régulation du sommeil paradoxal [Lu *et al.*, 2004, Fuller *et al.*, 2007, Brown *et al.*, 2008].

Afin de simuler des tracés d'activités neuronales par EEG, des modèles computationnels au niveau de la population neuronale ont été mis au point [Wilson et & Cowan, 1973, Lopes da Silva *et al.*, 1974]. En effet, la charge de calcul est trop importante lorsque les neurones sont considérés individuellement. Ces modèles ont montré un grand succès en générant des rythmes présents dans l'EEG d'éveil. Ainsi, la dynamique d'une population neuronale est décrite par une fréquence de décharge moyenne de tous les neurones de la population.

La transition éveil-sommeil NREM chez l'Homme a été décrite avec succès par un modèle, basé sur une commande de force homéostatique [Borbely, 1982, Daan *et al.*, 1984]. L'homéostasie est un phénomène par lequel un facteur clé est maintenu autour d'une valeur bénéfique pour le système considéré. Ce modèle est alors approfondi avec la mise en évidence de l'inhibition mutuelle entre les loci de l'éveil (le locus coeruleus (LC) et noyaux dorsal du raphé (DR)) et les populations favorisant le sommeil (noyau préoptique ventrolatéral (VLPO)) [Saper *et al.*, 2001, Diniz Behn et Booth, 2011]. En outre, cette transition est fortement influencée par le rythme circadien, agissant principalement par le biais du noyau supra-chiasmatique (SCN).

Des études ont montré que des populations de neurones, situées dans le tronc cérébral et l'hypothalamus, sont directement impliquées dans la régulation des états de sommeil et d'éveil [Moruzzi, 1972, McCarley et Hobson, 1975].

Au cours des 20 dernières années, plusieurs modèles mathématiques de réseau de régulation du type "sleep-wake" ont été proposés. Ces modèles sont principalement axés sur le sommeil des rongeurs

[Tamakawa *et al.*, 2006, Diniz Behn et Booth, 2011] et des humains [Tamakawa *et al.*, 2006, Phillips et Robinson, 2007, Rempe *et al.*, 2010].

En 2006, Tamakawa et son équipe [Tamakawa *et al.*, 2006] ont d'ailleurs montré qu'en effectuant de légères variations de paramètresA.1 sur la structure de leur réseau adapté au sommeil des rongeurs, ils arrivaient à produire un modèle de « sleep-wake » qualitativement similaires à ceux des humains.

De manière générale, il a été observé que les diverses équipes de recherche n'utilisent pas d'interface utilisateur graphique interagissant avec leurs modèles. Cela étant, certaines études comme celle de Joshi et son équipe [Joshi *et al.*, 2007] évoquent l'utilisation de MATLAB [1]. Ce dernier est un logiciel disposant d'une interface graphique et d'une multitude de fonctionnalités. Il permet donc, entre autres, la modification des paramètres de simulation mais aussi le traitement et l'analyse des données par le biais de matrices et de graphes.

Enfin, en lien avec ce dernier aspect, un groupe d'étudiants de Master 1 de Bio-Informatique de l'Université de Bordeaux [Darnige *et al.*, 2019] se sont basés sur le programme réalisé en 2016 par Costa et ses collègues [Schellenberger Costa *et al.*, 2016] pour établir un modèle computationnel pourvu d'une interface graphique. Ils ont traduit le programme original (`C++`) en `Python`. Les équations sont implantées de sorte à pouvoir héberger différents modèles. Actuellement, ce programme permet de réaliser des simulations basées sur des modèles humain comportant trois populations de neurones.

1.2 Les modèles mathématiques de régulation

1.2.1 Modèle à trois populations

La simulation du cycle éveil/sommeil intégrant trois populations neuronales est modélisée différentes équations différentielles. Elles sont similaires entre le modèle humain et le modèle rongeur, seuls les valeurs de certains paramètres varient. Chaque population neuronale va être nommée d'après l'état du cycle éveil/sommeil qu'elle promeut : *Wake, NREM* et *REM*. Le modèle et ses interactions sont représentés dans la figure 1.1. Tous les paramètres utilisés dans ces travaux sont présentés dans le tableau A.1.

- **La modélisation des fréquences de décharge neuronales par unité de temps :**

$$\frac{dF_W}{dt} = \frac{W_{max} \cdot (0.5 \cdot (1 + \tanh(\frac{I_W - \beta_W}{\alpha_W}))) - F_W}{\tau_W} \tag{1.1}$$

$$\frac{dF_N}{dt} = \frac{N_{max} \cdot (0.5 \cdot (1 + \tanh(\frac{I_N - \kappa_N \cdot H(t)}{\alpha_N}))) - F_N}{\tau_N} \tag{1.2}$$

$$\frac{dF_R}{dt} = \frac{R_{max} \cdot (0.5 \cdot (1 + \tanh(\frac{I_R - \beta_R}{\alpha_R}))) - F_R}{\tau_R} \tag{1.3}$$

  Ici, $\frac{dF_X}{dt}$ représente la fréquence des décharges neuronales par unité de temps. τ_x est la constante de temps. Wmax, Nmax, Rmax correspondent respectivement aux fréquences maximales des décharges des 3 populations (popWake, popNREM et popREM). Dans l'expression tanh(X), X représente la somme des stimulations.

- **Le processus d'homéostasie :**

$$\frac{dH}{dt} = \frac{H_{max} - H}{\tau_{hw}} \cdot \mathcal{H}(F_W - \theta_W) - \frac{H}{\tau_{hs}} \cdot \mathcal{H}(\theta_W - F_W) \tag{1.4}$$

1. `https://fr.mathworks.com/products/matlab.html`

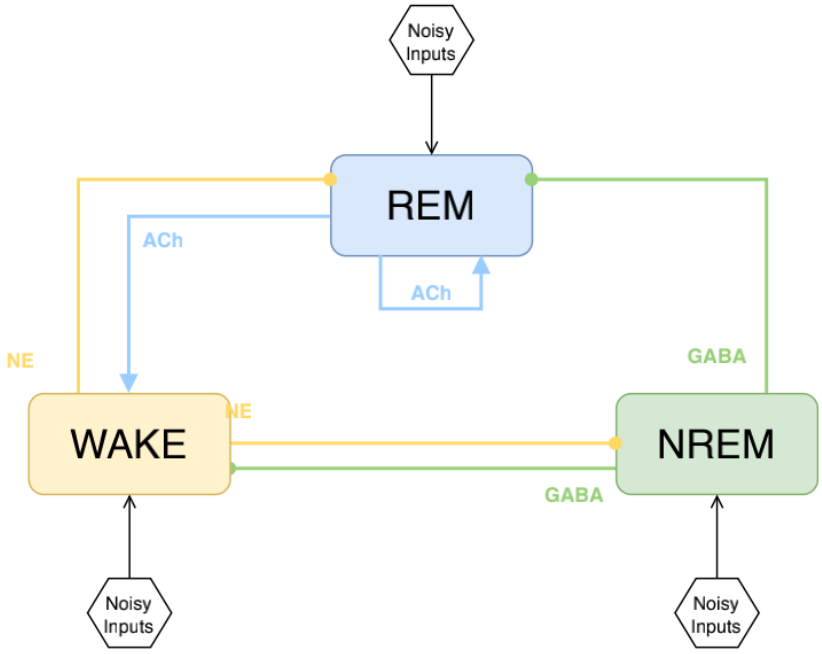


FIGURE 1.1 – **Schéma de l'architecture du modèle de 3 populations** Cette figure représente des populations neuronales impliquées dans les 3 états du cycle éveil-sommeil (*Wake, REM, NREM*). Les flèches modélisent une connexion dite excitatrice. Les segments à terminaison circulaire représentent les connexions inhibitrices. Les sigles NE (*noradrénaline*), ACh (*acétylcholine*) et GABA (*acide gamma-aminobutyrique*) sont les neurotransmetteurs excitateurs et inhibiteurs. Le bombardement synaptique est représenté par les formes grises : *Noisy inputs*. Il sera simulé par du bruit blanc Gaussien.

$Hmax$ est la force homéostatique maximale du système. $\mathcal{H}(x)$ représente la fonction de *Heaviside*, qui renvoie 0 si $x < 0$ et 1 si $x \geq 0$.

- **La modélisation de neuromodulateurs :**

Les concentrations des différents neurotransmetteurs sont représentées par les équations 1.5, 1.6 et 1.7. Elles dépendent des décharges neuronales (*firing rates*) précédemment calculés :

$$\frac{dC_{WX_e}}{dt} = \frac{tanh(\frac{F_W}{\gamma E}) - C_{WX_e}}{\tau E} \tag{1.5}$$

$$\frac{dC_{NX_e}}{dt} = \frac{tanh(\frac{F_N}{\gamma G}) - C_{NX_e}}{\tau G} \tag{1.6}$$

$$\frac{dC_{RX_e}}{dt} = \frac{tanh(\frac{F_R}{\gamma R}) - C_{RX_e}}{\tau R} \tag{1.7}$$

La concentration C de chaque neurotransmetteur dépend de l'activité synaptique F_W de la population produisant ce dernier (donc présynaptique).

1.2.2 Modèle à six populations

Sur la base du modèle à trois populations, les nouvelles recherches ont prouvé que le tronc cérébral et les populations neuronales de l'hypothalamus sont impliqués dans la régulation des états de sommeil et d'éveil. Les changements au niveau des neurotransmetteurs jouent un rôle important dans l'initiation et le maintien de la régulation éveil-sommeil.

L'éveil se caractérise par l'expression d'un taux élevé de noradrénaline (envoyé par le locus coeruleus (LC)), de sérotonine (envoyé par le raphé dorsal (DR)) et d'acétylcholine (emmit par le noyaux pontiques). Ceci y compris le tegmentum latérodorsal (LDT) et le pendunculopontine tegmentum (PPT) par la voie d'activation réticulaire ascendante vers les régions thalamocorticales. En revanche, les états de sommeil sont généralement caractérisés par une réduction des taux de tous ces neurotransmetteurs dans les régions du cerveau. Ceci à l'exception du taux d'acétylcholine qui redevient similaire à celui de l'état d'éveil pendant le sommeil à mouvements oculaires rapides (REM) [Fleshner *et al.*, 2011].

L'architecture de ce modèle est présenté dans la figure 1.2. Les paramètres en lien avec les équations qui le composent sont listées dans les tableaux A.2 et A.3. En triant toutes les relations, le modèle peut être grossièrement divisé comme suit :

- **La modélisation des fréquences de décharge neuronales par unité de temps :**

$$\frac{dF_{LC}}{dt} = \frac{F_{LC\infty}(g_{A,LC}C_A - g_{N,LC}C_N - g_{G,LC}C_G - g_{G(SCN),LC}C_{G(SCN)} + \eta(t)) - F_{LC}}{\tau_{LC}} \tag{1.8}$$

$$\frac{dF_{DR}}{dt} = \frac{F_{DR\infty}(g_{A,DR}C_A - g_{S,DR}C_S - g_{G,DR}C_G - g_{G(SCN),DR}C_{G(SCN)} + \eta(t)) - F_{DR}}{\tau_{DR}} \tag{1.9}$$

$$\frac{dF_{VLPO}}{dt} = \frac{F_{VLPO\infty}(g_{N,VLPO}C_N - g_{S,VLPO}C_S - g_{G,VLPO}C_G + g_{G(SCN),VLPO}C_{G(SCN)}) - F_{VLPO}}{\tau_{VLPO}} \tag{1.10}$$

$$\frac{dF_R}{dt} = \frac{F_{R\infty}(g_{A,R}C_A - g_{N,R}C_N - g_{S,R}C_S - g_{G,R}C_G - g_{G(SCN),R}C_{G(SCN)} + g_{PEP,R}C_{PEP}) - F_R}{\tau_R} \tag{1.11}$$

$$\frac{dF_{WR}}{dt} = \frac{F_{WR\infty}(g_{A,WR}C_A - g_{G,WR}C_G) - F_{WR}}{\tau_{WR}} \tag{1.12}$$

$$\frac{dF_{SCN}}{dt} = \frac{F_{SCN\infty}(CIRC(t) + SYN(t)) - F_{SCN}}{\tau_{SCN}} \tag{1.13}$$

$\frac{dF_X}{dt}$ représente les décharges par unité de temps. τ_x est la constante de temps. Le format $g_{i,X}$ décrit la force des synapses de la population X pour un neurotransmetteur i et C_i est la concentration de neurotransmetteur i. Dans $g_{G(SCN),X}$ et $C_{G(SCN)}$, le SCN est spécifié entre parenthèses pour identifier l'émetteur du GABA.

$F_{X\infty}$ et $F_{VLPO\infty}$ des équations précédentes (1.8, 1.9, 1.11, 1.13, 1.10 et 1.12) correspondent aux équations 1.14 et 1.15 :

$$F_{X\infty}(C) = X_{max} \cdot \left[0.5 \cdot \left(1 + \tanh\left(\frac{C - \beta_X}{\alpha_X}\right)\right)\right] \tag{1.14}$$

$$F_{VLPO\infty}(C) = VLPO_{max} \cdot \left[0.5 \cdot \left(1 + \tanh\left(\frac{(C - \beta_{VLPO}(h))}{\alpha_V LPO}\right)\right)\right] \tag{1.15}$$

X_{max}, $VLPO_{max}$ correspondent respectivement aux fréquences maximales de décharges pour

les 5 populations (*LC,DR,R,WR,SCN*) et par analogie à la fréquence maximale pour la population *VLPO*. $\frac{dF_{VLPO}}{dt}$ est obtenu par le processus d'homéostasie par l'équation $\beta_{VLPO}(h) = -k \cdot h$.

La population SCN est particulière puisqu'elle est régulée de manière circadienne.

- **La régulation circadienne :**
Pour modéliser la régulation circadienne, il existe 3 types de modèles : le modèle oscillateur à transmetteur unique (*STO, single transmitter oscillator model*, le modèle à double oscillateur à transmetteur-peptide (*TPDO, transmitter-peptide dual oscillator model*) et le modèle transmetteur-peptide colocalisé (*CTP, colocalized transmitter-peptide model*).
Seul le modèle STO sera utilisé dans ce projet car les autres incluent l'action d'un peptide dans la régulation du sommeil, ce qui ajoute un niveau de complexité inutile à ce stade de l'étude. Dans ce cas de figure, les forces synaptiques des populations LC, DR, VLPO et R pour un neurotransmetteur peptidique est alors nulle. L'entrée circadienne $CIRC(t)$ et l'entrée synaptique $SYN(t)$ sont respectivement calculées par les équations 1.16 et 1.17 :

$$CIRC(t) = \sin\left(\frac{2\pi t}{24 * 3600}\right) \tag{1.16}$$

$$SYN(t) = g_{A,SCN}(C_{A(R)} + C_{A(WR)}) + g_{S,SCN}C_S \tag{1.17}$$

- **Le processus d'homéostasie :**

$$\frac{dh}{dt} = \mathcal{H}[(F_{LC} + F_{DR}) - \theta_W] \cdot \frac{(H_{max} - h)}{\tau_{hw}} - \mathcal{H}[\theta_W - (F_{LC} + F_{DR})] \cdot \frac{h}{\tau_{hs}} \tag{1.18}$$

$Hmax$ est la force homéostatique maximale du système. $\mathcal{H}(x)$ représente la fonction *Heaviside*, qui renvoie 0 si $x < 0$ et 1 si $x \geq 0$.

- **La modélisation des neuromodulateurs :**

Les concentrations des différents neurotransmetteurs sont représentées par les équations 1.19 et 1.20. Elles dépendent des décharges neuronales (*firing rates*) des différentes populations (*LC, DR, VLPO, R, WR, SCN*), précédemment calculées dans les équations 1.8, 1.9, 1.10, 1.11, 1.12 et 1.13 :

$$\frac{dC_i}{dt} = \frac{C_{i\infty}(F_{pop}) - C_i}{\tau_i} \tag{1.19}$$

$$C_{i\infty}(F_{pop}) = \sigma_i(t) \tanh(\frac{F_{pop}}{\gamma_i}) \tag{1.20}$$

Ici, $\sigma_i(t)$ correspond à l'amplitude utilisée pour décrire le bruit. Les relations correspondantes entre *pop* et *i* (pop génère i) sont respectivement présentées dans la figure 1.2.

- **Architecture du modèle**

La figure 1.2 montre l'architecture du réseau à 6 populations neuronales. Le schéma a été réalisé et adapté d'après l'article de Fleshner et ses collègues [Fleshner *et al.*, 2011].

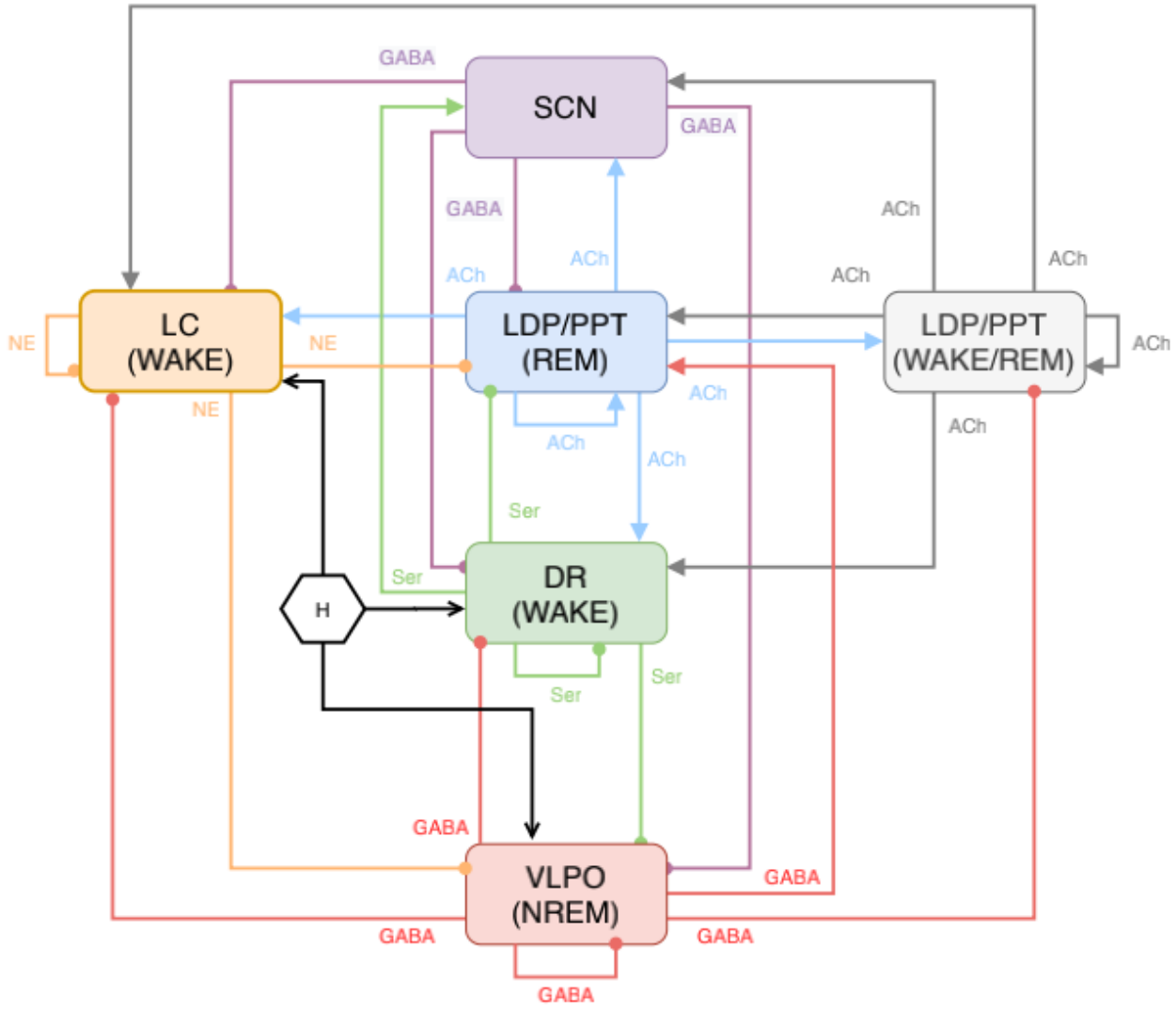


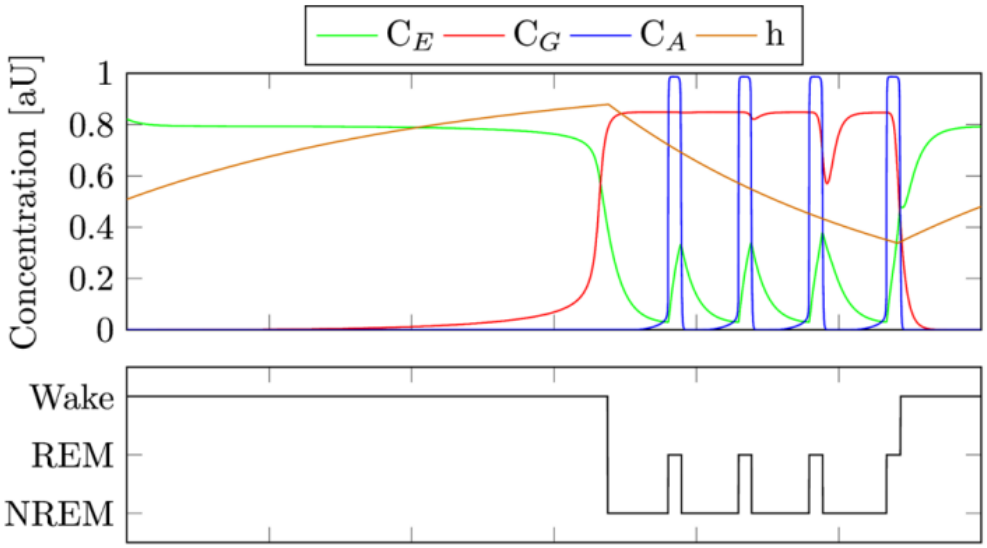
FIGURE 1.2 – **Architecture du modèle à 6 populations neuronales**. Les flèches représentent les connexions excitatrices et les segments terminés d'un cercle rempli représentent les connexions inhibitrices. `H` représente la force homéostatique. Lorsque le réseau est en état d'éveil, les populations LC et DR sont activées et sécrètent respectivement de la noradrénaline (NE) et de la sérotonine(Ser). Ces deux neurotransmetteurs inhibent les neurones des régions VLPO, LDP / PPT. C'est cette activité qui induit l'état d'éveil. Dans la phase NREM, VLPO semble jouer un rôle important. L'acide gamma-aminobutyrique(GABA) qu'elle sécrète inhibe la phase REM/WAKE. Dans la phase REM, l'acétylcholine (ACh) sécrétée par LDP / PPT peut activer la population WAKE. En d'autres termes, l'état d'éveil n'est possible qu'au sein d'une phase de REM mais pas directement dans une phase de NREM. De plus, bien que le SCN ne soit pas le principal acteur de tous les états (Wake, REM, NREM), l'ACh et la Ser qui lui parvient pendant l'état d'éveil peuvent alors stimuler la synergie de sécrétion de GABA, neurotransmetteurs favorisant le sommeil. Enfin, l'effet inhibiteur mutuel entre le mécanisme éveil-sommeil et le mécanisme d'endormissement forme finalement "l'interrupteur éveil-sommeil".

1.3 Programme de Costa et ses collègues (2016)

1.3.1 Outils et Fonctionnement

Le programme de Costa et ses collègues [Schellenberger Costa *et al.*, 2016] incluent neuf fichiers en langage C++ et un fichier en MATLAB. Le résultat de cette combinaison de fichiers est l'obtention d'un graphique tel que sur la figure 1.3.

Il existe deux fichiers principaux : le fichier `<Cortex_SR.cpp>` qui contient la fonction d'entrée



.

FIGURE 1.3 – **Visualisation graphique des résultats attendus** La partie haute du graphique représente l'évolution de la concentration des neurotransmetteurs au cours du temps du modèle à 3 populations humain. La partie basse modélise l'hypnogramme permettant de connaître l'état du réseau, toujours au cours du temps

`main()` et le fichier `<Cortex_SR_mex.cpp>` qui est le fichier source de MEX. Le fichier MEX est un fichier binaire dérivé du langage C. Il s'agit d'un programme dynamique qui peut être chargé et exécuté automatiquement par l'interrogateur MATLAB. Les interactions d'inclusion et d'utilisation entre les différents fichiers ainsi que leur positionnement algorithmique seront présentés dans le diagramme 1.6 de la partie suivante. Les paramètres et les fonctions de chaque fichier sont résumés dans le tableau 1.4.

Fichiers	Type du fichier	Nombre de lignes	Nombre de fonctions	Nombre de paramètres	Structures de données	Lié à une classe	Utilité
`Random_Stream.h`	fichier d'en-tête	56	6	3	`int`	oui	Génération du bruit
`Cortical_Column.h`	fichier d'en-tête	197	22	51	`vector`	oui	Déclaration des paramètres
`Cortical_Column.cpp`	fichier en C++	178	16	13	`vector`	oui	Initialisation des populations
`Sleep_Regulation.h`	fichier d'en-tête	132	9	37	`vector`	oui	Déclaration des paramètres
`Sleep_Regulation.cpp`	fichier en C++	78	6	7	`vector`	oui	Initialisation des populations
`ODE.h` (***Ordinary Equation Solver***)	fichier d'en-tête	42	1	0	`vector`	oui	Résolution des équations différentielles
`Cortex_SP.cpp`	fichier en C++	74	1	4	`vector`	oui	Point d'entrée
`Data_Storage.h`	fichier d'en-tête	49	1	11	`vector`	oui	Stockage des données
`Cortex_SR_mex.cpp`	fichier en C++	129	2	11	`mexArray`	oui	Fichier source pour MEX
`Plots.m`	fichier en MATLAB	98	-	11	`mexArray`	-	Porduction des graphes

FIGURE 1.4 – Tableau d'analyse des fichiers

Voici le détail de chaque fichier :

Les fichiers d'en-tête, avec des noms se terminant par `.h`, permettent de partager des déclarations entre plusieurs fichiers composants un même programme.

`#pragma once` est une directive pré-processeur qui assure que le fichier source actuel ne peut être inclus qu'une seule fois dans la compilation.

`#include <vector>` et `#include <cmath>` se chargent respectivement d'introduire un module de vecteur ainsi qu'un groupe de fonctions mathématiques.

Le fichier `<Sleep_Regulation.h>` déclare une classe `Sleep_Regulation` qui comporte 9 fonctions et 37 variables. Il contient aussi une classe `Cortical_Column` qui sera expliquée plus bas. Parmi ces déclarations, il y a deux fonctions basiques qui sont exécutées dans toutes les parties. La première fonction, `inline std::vector<double> init (double value)`, permet de retourner un vecteur pour exprimer toutes les variables qui facilitent le calcul de la valeur différentielle. La deuxième fonction, `inline void add_RK (std::vector <double>& var)`, montre une équation différentielle par la méthode d'intégration Runge-Kutta d'ordre 4 (*RK4*) [Baty, 2018]. Les autres paramètres et fonctions sont utilisés dans le fichier suivant pour établir les formules décrivant les relations établies entre les 3 populations.

Un fichier d'en-tête est souvent inclus par une directive pré-processeur telle que `#include "Sleep_Regulation.h"`, dans un fichier définition portant le même nom mais dans un format différent `.cpp`.

Le fichier `<Sleep_Regulation.cpp>` propose six fonctions importantes [Diniz Behn et Booth, 2012] afin d'obtenir les fréquences de décharge neuronale selon les divers états d'éveil (Wake,(F_W), NREM (F_N) et REM (F_R)), les concentrations des différents neurotransmetteurs (noradrénaline (C_E), acétylcholine (C_A) et GABA (acide gamma-aminobutyrique, C_G)) et l'homéostasie (H). Ces valeurs varient par unité de temps.

Chaque population neuronale possède une fonction d'entrée qui représente la somme des poids post-synaptiques (connexions "entrantes") et la concentration en neurotransmetteur de ces synapses. Les poids synaptiques sont modélisés par la variable g_{XY} où X est la population pré-synaptique et Y est la population post-synaptique.

Les trois dernières fonctions se chargent d'exprimer les 7 équations principales avec comme pas de temps dt.
Tous ces paramètres, obtenus par les fonctions et ses changements au cours du temps, vont être enregistrés dans le fichier `Data_Storage.h` sous la forme `Array`.

Le fichier `<Random_Stream.h>` comporte deux classes générant respectivement une série de nombres aléatoires suivant une loi normale et une série de nombres aléatoires suivant la loi uniforme.

`#include <random>` est une directive d'inclusion pour introduire le module de génération de nombres aléatoires.

Sur la base des fichiers présentés ci-dessus, le fichier `<Cortical_Column.h>` déclare une classe `Cortical_Column` (citée dans le fichier `Sleep_Regulation.h`) qui comporte 22 fonctions et 51 paramètres.

`connect_SR(Sleep_Regulation& SR_Network)` permet la liaison avec `Sleep_Regulation`.De par cette fonction, tous les paramètres de `Sleep_Regulation.h` sont appelés.
La fonction `Update_Bifurcation_Parameters (void)` permet alors de lier les deux grandes parties.

`std::vector<randomStreamNormal> MTRands;` déclare une variable `MTRands` représentant le bruit régi par la loi normale (*Noisy Input*).

Tous les paramètres utilisés pour les calculs sont également stockés dans des vecteurs.

Le fichier `<Cortical_Column.cpp>` est établi selon l'article [Schellenberger Costa *et al.*, 2016] qui se charge de lister toutes les fonctions nécessaires à l'expression de Vp (ou Vi). Vp est donc l'évolution du voltage de la membrane par unité de temps au sein des populations *"Pyramidal"* et *"Inhibitory"*.
Il s'y trouve deux principales équations qui ne seront pas détaillées puisqu'elles ne sont pas traitées dans ce projet.

Les paramètres obtenus par les fonctions et ses changements au cours du temps sont enregistrés dans le fichier `Data_Storage.h`. La méthode RK4 est toujours utilisée ici pour calculer les changements sur une courte période.

Le fichier `<ODE.h>` ne contient qu'une fonction qui permet de combiner les deux grandes parties `Sleep_Regulation` et `Cortical_Column`. ODE permet de résoudre numériquement les équations différentielles selon les formules précédemment citées à chaque pas de temps.

Comme cette partie (`<Cortex_SR.cpp>`) est le point d'entrée, elle ne contient qu'une seule fonction `int main(void)`. Cette fonction initialise les deux grandes populations `Sleep_Regulation` et `Cortical_Column` puis appelle la fonction `Cortex.connect_SR(SR)` pour connecter les deux parties. Ensuite, avec l'aide de la boucle du `ODE(Cortex, SR)` au cours du temps, la fonction effectue la simulation.

Dans le fichier `<Data_Storage.h>`, il y a de nouveau la fonction `get_data()`. Les données obtenues par toutes les parties sont stockées dans le tableau détaillé ci après :

`std::vector<double> pData` inclut toutes les valeurs des variables à chaque pas de temps et pour toute la durée de la simulation.

`pData[0][counter] = Col.Vp [0];` Vp est le voltage membranaire de la population de neurones pyramidaux.

`pData[1][counter] = Col.Vi [0];` Vi est le voltage membranaire de la population de neurones inhibiteurs (interneurones).

`pData[2][counter] = SR.f_W [0];` f_W est la décharge neuronale de la population d'éveil.

`pData[3][counter] = SR.f_N [0];` f_N est la décharge neuronale de la population NREM.

`pData[4][counter] = SR.f_R [0];` f_R est la décharge neuronale du REM.

`pData[5][counter] = SR.C_E [0];` C_E est la concentration de noradrénaline.

`pData[6][counter] = SR.C_G [0];` C_G est la concentration de GABA.

`pData[7][counter] = SR.C_A [0];` C_A est la concentration d'acétylcholine.

`pData[8][counter] = SR.h [0];` h est l'homéostasie.

`pData[9][counter] = Col.g_KNa [0];` g_KNa est la force des synapses sodium-potassium.

`pData[10][counter]= Col.sigma_p [0];` *sigma_p gain neuronal.*

La dernière partie du fichier `<Cortex_SR_mex.cpp>` se charge de déclarer les paramètres du graphe et d'initialiser un fichier MATLAB pour tracer les courbes comme dans la figure 1.3 qui représente l'activité corticale. Ce fichier peut être compilé avec *Matlab Executable(MEX)* pour générer un fichier en format `.mex`. Un fichier source de MEX en langage C++ ainsi que celui-ci, il existe toujours une directive `#include "mex.h"`. Dans ce fichier, la fonction d'entrée `main()` est changée par une fonction définie suivante :

```
mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
```

mxArray est la structure des données définie par `mex.h`, *nlhs* est le nombre des variables de sortie, *plhs* est le vecteur des pointeurs des variables de sortie, *nrhs* est le nombre des variables d'entrée, *prhs* est le vecteur des pointeurs des variables d'entrée.

Par conséquent, cette fonction peut stocker toutes les variables de sortie et les variables d'entrée après la boucle de `ODE` dans le `mxArray`.
Afin de faire fonctionner ce fichier dans MATLAB, son fichier compilé est obligatoirement nommé comme `mexFunction`.

Enfin, le fichier `<Plots.m>` peut extraire les données préparées dans `mxArray` pour dessiner le graphe (plot) dans MATLAB. Les variables de l'axe X et l'axe Y de chaque plot sont fixés par ses codes.

1.3.2 Architecture et diagramme des classes

Les figures 1.5 et 1.6 illustrent respectivement l'architecture et les classes du programme de Costa et ses collègues [Schellenberger Costa *et al.*, 2016].

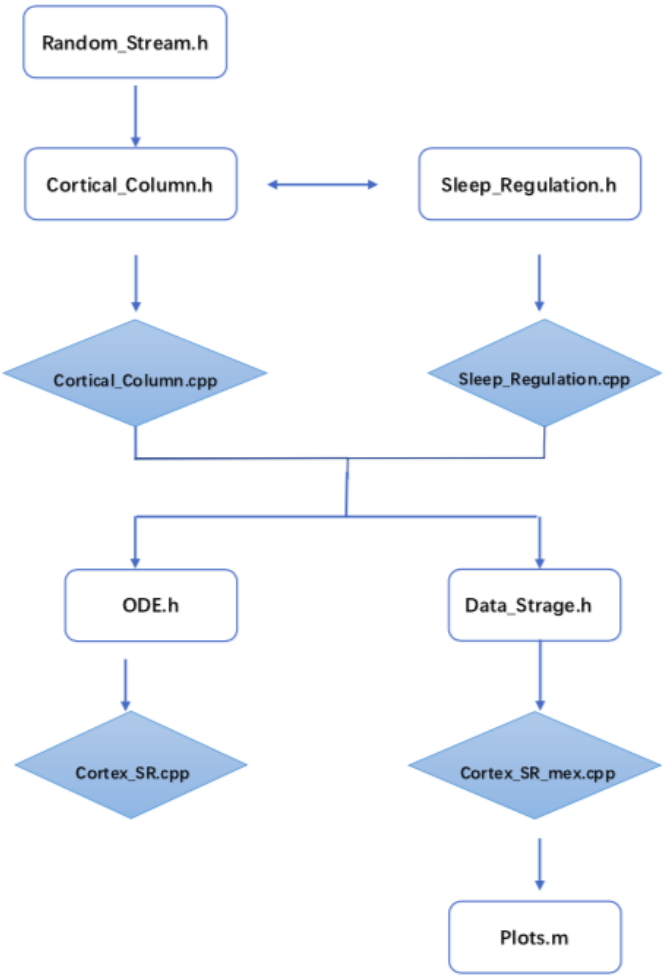


FIGURE 1.5 – **Architecture du programme** Dans cette figure, il y a trois fichiers `header` comportant trois classes du même nom. Le ficher `Cortical_Column.cpp` utilise les fonctions viennent de `Cortical_Column.h`. Il en est de même pour les autres fichiers (`Sleep_Regulation.cpp, Cortex_SR.cpp et Cortex_SR_mex.cpp`). Le sens de la flèche indique que les fonctions sont référencée. Au final, `Cortex_SR_mex.cpp` regroupe toutes les fonctions et collecte toutes les données. Le fichier `plots.m` est en MATLAB.

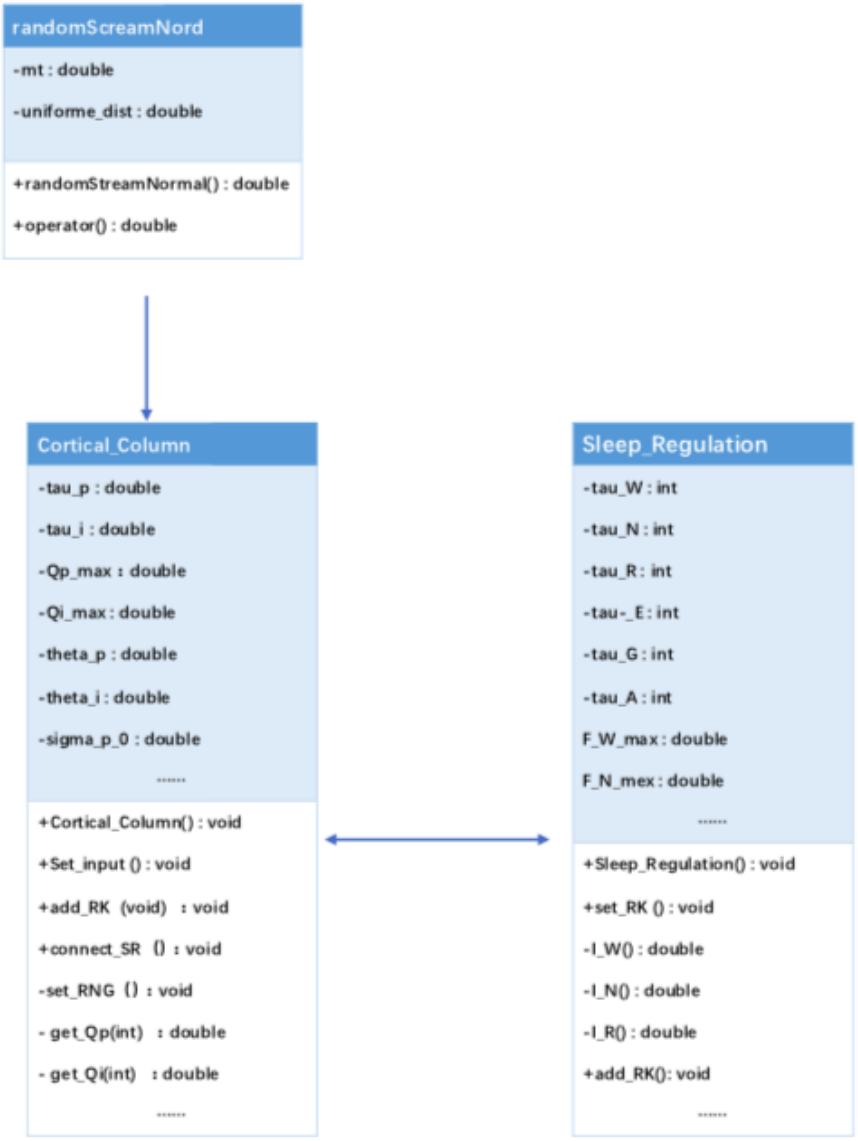


FIGURE 1.6 – **Diagramme des classes** Cette figure représente les trois classes qui ont été utilisées ainsi que leurs interactions. La classe `randomStreamNormal` est dans le fichier `Random_Stream.h`. Elle est utilisée dans `Column_Column.h`. La classe `Cortical_Column` et la classe `Sleep_Regulation` utilisent les données de l'une de l'autre.

1.4 Programme des Master 1 de Bordeaux (2019)

1.4.1 Outils et Fonctionnement

Ce programme est basé sur le programme de Costa [Schellenberger Costa *et al.*, 2016] présenté précédemment mais a été traduit en Python et en R par un groupe d'étudiants en bioinformatique de l'université de Bordeaux [Darnige *et al.*, 2019]. En effet, ces langages étaient privilégiés par l'équipe de recherche à l'origine de ce projet. Leur programme comporte six fichiers dont cinq en Python et un en R.

La version 3 de Python permet la programmation fonctionnelle, orientée objet et structurale. Plusieurs librairies ont été ajoutées afin de permettre l'ajout de certaines fonctionnalités. Elles seront présentées au fur et à mesure de l'analyse du code.

Le programme est articulé entre différents fichiers qui alimentent le fichier `Main.py`. Ce dernier permet d'initialiser l'interface et le modèle pour les différentes fonctionnalités disponibles. Il offre alors une interface claire, avec une prise en main facile, directement dédiée à l'utilisateur avec notamment

des menus dont les différentes sections sont choisies par des boutons. L'interface a été réalisée à l'aide de la librairie `tkinter`. Une courte fonction (`doStats()`) permet de lancer le script R pour réaliser les statistiques directement depuis l'interface.

La fonction `loadModel()` utilise quatre dictionnaires obtenus par la fonction `read_parameters()` importée depuis `manage_parameters.py` au début du fichier. Cette dernière lit un fichier `.txt`, en extrait tous les paramètres et les sauvegarde dans un dictionnaire de dictionnaire.

Le fichier `manage_parameters.py` peut également sauvegarder tous les paramètres utilisés pendant la simulation dans un autre fichier `.txt` grâce à la fonction `write_parameters()`. Cela permet à l'utilisateur de modifier les paramètres du modèle chargé directement depuis l'interface et d'obtenir un fichier `.txt` formalisé avec les nouveaux paramètres saisis. Ce dernier pourra alors être chargé directement lors d'une autre simulation.

`GUI.py` est le fichier python qui gère toute l'interface graphique. Il ne possède qu'une seule classe avec différentes fonctions permettant l'affichage des boites de dialogues, des paramètres, et de toutes les fonctions qui permettent au programme de fonctionner. Ici, c'est principalement la librairie `tkinter` qui est utilisée. Les interactions possibles entre le programme et l'utilisateur passe par ce fichier.

Le fichier `SleepregulationOOP` est le fichier principal de ce programme. Il est divisé en plusieurs classes :

- `Network` : permettant l'initialisation de la classe `Network`. Cette classe est utilisée pour gérer la simulation. Elle propose une relation d'agrégation avec la classe `NeuronalPopulation`, la classe `HomeostaticSleepDrive`, la classe `Connection` et la classe `Injection`. Cela permet à tous les paramètres d'être appelés dans l'interface graphique. Elle contient plusieurs fonctions, notamment pour l'initialisation, la création du bruit blanc gaussien, la mise en place des paramètres, le lancement de la simulation, la création des hypnogrammes, l'écriture des résultats dans un fichier, l'enregistrement, les méthodes de modification du réseau, et le débuggage.
- `NeuronalPopulation` : Classe permettant la représentation d'une population neuronale. La population neuronale est stockée sous forme d'un dictionnaire et différentes fonctionnalités peuvent s'y appliquer en lien avec la concentration de neurotransmetteur et leur bombardement sur la population.
- `HomeostaticSleepDrive` : Celle-ci permet une simulation du processus homéostatique intervenant dans la régulation du cycle sommeil/éveil par la création d'objets à comportement cyclique.
- `Connection` :Cette classe permet de générer des connexions entre les différentes populations de neurones. Selon différentes conditions de connexion, cette classe propose respectivement 5 modes : `NP-NP` décrit une connexion entre une population neuronale et une autre population neuronale qui va calculer le produit de la concentration en neurotransmetteur pré-synaptique avec la force des synapses post-synaptique. `HSD-NP` décrit une connexion entre le processus d'homéostasie et la population neuronale associée qui va calculer le produit de l'homéostasie avec un coefficient constant. `NP-HSD` décrit une connexion entre la population neuronale et le processus d'homéostasie qui génère les fréquences des décharges de cette population neuronale. `NP-MIE-NP` décrit une connexion entre une population neuronale et une autre population neuronale en rajoutant une injection de neurotransmetteur agoniste. Elle va calculer la concentration de neurotransmetteur ayant une action excitatrice. `NP-MII-NP` décrit une connexion entre une population neuronale et une autre population neuronale en rajoutant une injection de neurotransmetteur antagoniste. Elle va calculer la concentration de neurotransmetteur ayant une action inhibitrice.
- `Injection` : Cette dernière classe gère la nature des simulations de micro-injections de neurotransmetteurs.

Le fichier `graphic.py` contient toutes les fonctions utilisées pour l'affichage des graphiques. Il utilise `matplotlib`, `graphviz` et `tkinter` principalement. Les fonctions permettent l'affichage des fenêtres, leur aménagement, la création du graphique, des graphiques de moyennes, la comparaison avec les contrôles, la lecture des fichiers `.csv` et la création de graphiques qui en sont issus.

Le fichier `Stats.r` est rédigé en langage R. Ce dernier est un langage propre au domaine des statistiques. Ici, il est utilisé pour faire différents tests et calculs statistiques mais aussi pour la représentation graphique des résultats obtenus à l'issue de ces dits calculs.

Les librairies `Numpy` et `math` sont utilisées pour la manipulation des données et des équations. Quant à `csv`, elle s'avère très utile pour la sauvegarde des résultats des simulations et la réalisation des graphiques.

Le tableau de la figure 1.7 présente un récapitulatif des fichiers composant le programme.

Fichiers	Nombre de lignes	Nombre de fonctions	Nombre de classes	Utilité
`Main.py`	150	2	-	Initialise le modèle. Exécute le script `Stats.R` Initialise la fenêtre principale de l'interface avec les menus
`GUI.py`	511	28	1	Initialise les fenêtres, les sous fenêtres de l'interface et les menus déroulants. Afficher les données à l'utilisateur Récupère et sauvegarde les modifications de l'utilisateur
`SleepRegulationOOP.py`	509	49	5	Initialise et gère les simulations. Crée les populations neuronales et les connexions à l'aide des paramètres du fichier `.txt`. Contient les équations. Sauvegarde les paramètres modifiés par l'utilisateur. Permet la modélisation de micro-injections. Utilise deux méthodes de résolution d'équations : RK4 et Euler.
`manage_parameters.py`	81	2	-	Récupère les paramètres du fichier .txt pour les stocker dans des dictionnaires. Sauvegarde les paramètres saisis par l'utilisateur dans l'interface dans un fichier `.txt`.
`graphic.py`	440	9	-	Lit les données issues des simulations. Produit les graphes, graphes moyens et écarts types.
`Stats.r`	288	-	-	Importe les données issues des simulations. Réalise les tests et calculs statistiques (ANOVA, écart type…). Produit les graphes issus de ces tests. Sauvegarde les résultats dans des fichiers.

FIGURE 1.7 – Tableau récapitulatif du programme

Les liens entre le code C++ et Python sont nombreux même si l'organisation est parfois un peu différente au sein des programmes. Les fonctions remplies restent très conformes. La figure 1.8 réalisée par les étudiants de 2019 [Darnige *et al.*, 2019], résume les différences notables entre le programme initial de 2016 [Schellenberger Costa *et al.*, 2016] et leur programme.

Le premier fichier `CortexSR.CPP` joue le même rôle que `main.py`. En effet il est le fichier principal pour les tests de compilation et d'exécution des tests, avec des choix de simulation tout à fait similaire au fichier python.

Le second fichier `CorsterSRmex.cpp`, permet l'analyse statistique des simulations. Il s'accorderait donc avec les fichiers `Stats.r` ainsi que `graphic.py`. Ces derniers fichiers sont tout de même plus complets dans le cadre d'analyses descriptives et c'est pour cette raison que le `.cpp` fait appel à un fichier `plots.m` pour les représentations graphiques.

Le troisième fichier `CorticalColumn.cpp` est structurellement très apparenté avec `SleepRegulationOOP.py` du programme python. Effectivement il permet lui aussi la modélisation des populations neuronales, des connexions entre elles et des différentes stimulations par les divers neurotransmetteurs. Les multiples régulations par les neurotransmetteurs sont elles aussi représentées sous forme d'itérations.

Le dernier fichier `Sleepregulation.cpp` tient lui aussi un équivalent python mais intégré dans le `SleepRegulationOOP.py` mentionné précédemment. En effet, il est comparable à la classe `HomeostaticSleepDrive` puisqu'il modélise les cycles naturels du sommeil.

Attribute	Costa *et al.*, 2016	Darnige, Grimaud, Gruel, Kuntz, 2019
Languages used	C++, Matlab	Python 3, R
Class organization	1 super class	Biological compartments representing neuronal populations. Class representing synaptic connections
ODE methods	Runge-Kutta 4th order	Runge-Kutta 4th order or Euler
Number of Populations	3	1-infinity
Noise	White noise input in the cortical submoduleone pyramidal and inhibitory populations	Additive white Gaussian noise [Hz] incorporated in the next step firing rate equation
Microinjections	Not included	Microinjection of agonist or antagonist neurotransmitters in a target population possible
Lesions	Not included	Lesions between two neuronal populations can be simulated
Parameters	Modified within script	Modified in GUI, or modified in parameters text file
Graphical User Interface	None	Tkinter
Visualization	Neuromodulator concentrations, hypnogram	Neuromodulator concentrations, firing rates, hypnogram. Option to view means and standard deviations of multiple results
Statistics	Analyzed seperately	Automated one-way ANOVA, Tukey HSD. Bar graphs of mean results by state

FIGURE 1.8 – **Tableau comparatif des programmes [Darnige *et al.*, 2019]**. Ce tableau présente un récapitulatif des fonctionnalités en comparaison entre le programme initial de 2016 [Schellenberger Costa *et al.*, 2016] et le programme réalisé en 2019 par les étudiants de Bordeaux.

Finalement les fonctions de `manage_parameters.py` sont remplies par le fichier `Cortex.cpp`. Une autre différence notable est que de nombreux fichier `.h` sont associés au code `.cpp` afin de faciliter les modifications du code source.

1.4.2 Architecture et diagramme des classes

Les figures 1.9 et 1.10 sont extraites du rapport de projet des étudiants de 2019 [Darnige *et al.*, 2019]. Elles présentent respectivement l'architecture et les classes de leur programme.

1.5 Conception

1.5.1 Besoins et Objectifs

Le client possède l'application créée en 2019 par le groupe d'étudiants de Bordeaux [Darnige *et al.*, 2019], et dont les fonctionnalités permettent d'effectuer des simulations quant à la dynamique neuronale du sommeil. Cette année, nous avons été sollicités afin d'en améliorer le programme et de le

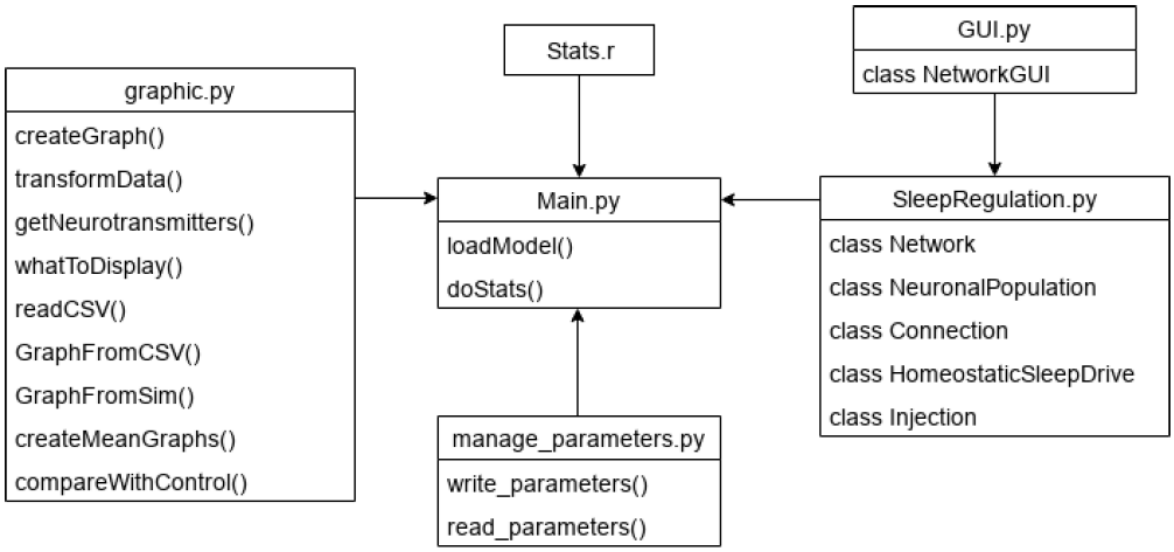


FIGURE 1.9 – **L'architecture du programme de 2019 [Darnige *et al.*, 2019]**. Cette figure représente l'architecture du programme réalisé en 2019 par les étudiants de Bordeaux. Les flèches montrent les interactions entre les différents fichiers qui le constituent. Pour chaque fichier, la liste des fonctions est représentée.

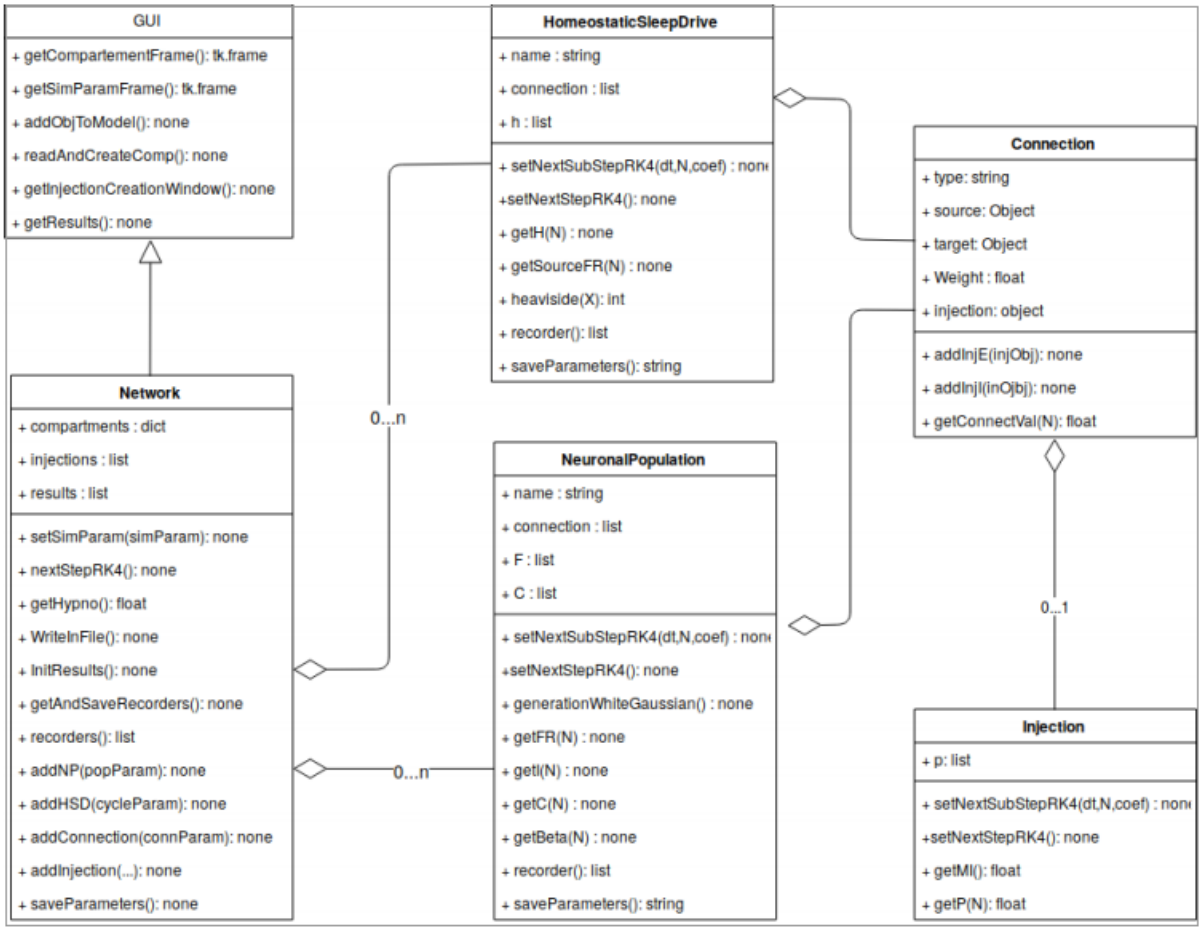


FIGURE 1.10 – **Diagramme des classes 2019 [Darnige *et al.*, 2019]**. Ce diagramme représente les différentes classes et leurs interactions au sein du programme des étudiants de 2019 [Darnige *et al.*, 2019]. Les variables et les fonctions y sont également présentées.

rendre plus polyvalent. En effet ce dernier, tel qu'il a été conçu, ne permet de charger que des modèles humains. Il doit désormais être capable d'effectuer des simulations sur des rongeurs. Des modifications y seront alors nécessaires. Dans un premier temps, il sera question de comprendre le programme réalisé en 2019 pour, dans un deuxième temps, y intégrer nos nouveaux modèles. Ils simuleront la dynamique du sommeil chez les rongeurs avec des représentations à 3 et 6 populations de neurones. Ces modèles pourront connaître des variantes afin de permettre la simulation de lésions ou d'injections de neurotransmetteurs sur des populations cibles.

Les objectifs principaux sont donc de créer deux modèles : un modèle rongeur à trois populations de neurones [Diniz Behn et Booth, 2012] ainsi qu'un modèle rongeur à six populations, qui tient compte du cycle circadien, comme l'ont fait Fleshner et son équipe [Fleshner *et al.*, 2011]. Les objectifs secondaires seront de procéder à des tests de lésions et de micro-injections, basés sur les modèles ainsi créés [Diniz Behn et Booth, 2010]. Afin de s'assurer de la reproductibilité des résultats, il sera nécessaire d'effectuer dix simulations pour chaque modalité expérimentale pour ensuite en tirer des graphes moyens, accompagnés de tests statistiques. Ces analyses permettront alors de démontrer que le programme est fiable et fonctionnel. Afin de garantir une certaine conformité des modèles, les graphiques devront visuellement se rapprocher au maximum de ceux retrouvés dans les divers articles utilisés pour ce projet. Enfin, il est également entendu que l'optimisation de l'existant fait partie des objectifs.

1.5.2 Amélioration de l'existant

Pour implanter le modèle rongeur à 3 populations de neurones, il s'agira d'intégrations au sein des fichiers plutôt faibles. Pour les autres modèles, plus complexes, il sera question d'intégrations plus fortes, notamment dans le fichier `SleepRegulationOOP.py` qui comporte l'ensemble des équations.

Selon les informations tirées des articles scientifiques, les équations ne changent pas au sein des modèles à trois populations en fonction du type de sujet[Diniz Behn et Booth, 2012]. Par conséquent afin de créer le modèle rongeur à 3 populations de neurones, il s'agira de créer un nouveau fichier `.txt` avec des valeurs de paramètres adaptées. Pour rester conforme aux modèles déjà validés par la communauté scientifique, il sera impératif de s'appuyer sur les articles mis à disposition.

De la même manière, pour le modèle à 6 populations, il s'agira de créer un autre fichier `.txt` contenant des paramètres adaptés. Cependant, il faudra aussi ajouter et modifier des équations au sein du programme pour que celui-ci puisse s'adapter au type de modèle chargé. La visualisation graphique sera elle aussi changée afin de représenter au mieux les 6 populations.
Plus généralement, l'échelle de temps des représentations graphiques sera mis a jour selon le type de sujet expérimenté (en heure pour le modèle humain et en minutes pour les modèles rongeur).

Dans le cadre d'un approfondissement des simulations, les tests lésionnels et les injections de neurotransmetteurs devront être effectifs sur chacun des modèles. En ce qui concerne cette partie au niveau de l'interface graphique, la fenêtre permettant de gérer les injections devra être améliorée afin d'afficher le type d'injection après sa sélection.

Quant au script R, il présente des commandes obsolètes qui devront être corrigées.

Pour chacune de ces étapes, il sera important de constamment s'assurer que le programme ne présente aucune erreur.

Enfin, l'interface présente quelques bugs d'affichage (taille variant à chaque clic de bouton, fenêtre des paramètres qui ne s'affiche pas correctement, etc.). Il sera donc important de l'améliorer. Il en est de même pour l'affichage des graphiques qui n'est pas optimal (légendes se positionnant sur les courbes, dernière valeur de temps absente, couleurs trop ressemblantes, etc.)

Chapitre 2

Réalisation

2.1 Implémentation

2.1.1 Interface

L'interface graphique, réalisée en 2019, permet de pratiquer sur l'ensemble des simulations basées sur les modèles rongeurs élaborés cette année. En effet, du chargement des fichiers `.txt` à la visualisation des résultats, les simulations se réalisent normalement de la même manière que pour les modèles humains. Cependant, cette application révèle quelques défauts de présentation, notamment lorsqu'il s'agit de visualiser certains paramètres. Ainsi, les modifications apportées sur l'interface se résument à quelques améliorations faites au niveau de l'affichage. Bien que cela puisse sembler purement esthétique, les défauts relevés se sont montrés inconfortables pour l'utilisateur et, dans le pire des cas, ils empêchaient la visualisation de certaines valeurs de paramètres. Il s'est alors avéré primordiale de les corriger.

La première anomalie remarquée sur cette interface est la taille de sa fenêtre principale et celles de ses fenêtres satellites. En effet, leurs dimensions étaient figées et, par conséquent, n'étaient pas adaptées à toutes les tailles d'écran. Par exemple, si ce dernier était trop petit, certaines fenêtres pouvaient sortir du cadre visible et l'utilisateur devait se charger lui-même de redimensionner les fenêtres avant de pouvoir se servir pleinement de l'interface ou de visualiser correctement les résultats. Le client a aussi fait remarquer que le fait d'attribuer des tailles différentes pour chacune des fenêtres rendait désagréable les manipulations. Afin de pallier ces deux problèmes simultanément, le programme récupère désormais la taille de l'écran sur lequel il est lancé et les dimensions de l'application s'adaptent de manière automatique chez tous les utilisateurs. Au niveau du code, cela se traduit par l'utilisation des méthodes `tkinter` `window.winfo_screenwidth()` et `window.winfo_screenheight()` qui récupèrent respectivement la largeur et la hauteur de l'écran sous la forme d'entiers. Ces valeurs sont ensuite stockées dans des variables. Ainsi, les paramètres dimensionnels de la fenêtre `window` et de son `Notebook` sont corrigés avec ces nouvelles variables au sein du fichier `main.py`. De la même manière, la taille la fenêtre `Display Compartments Parameters)`, ou plus exactement la taille du canevas sur lequel sont représentés les compartiments biologiques est ajustée grâce à ces méthodes.

La deuxième amélioration à apporter à l'interface graphique concernait l'affichage des compartiments, correspondants à la visualisation des populations neuronales associées à leurs paramètres respectifs. Cette fonctionnalité est générée à partir du bouton `Display Compartments Parameters` de l'onglet `Parameters`. Ici, lorsque le modèle rongeur à 6 populations de neurones était chargé, l'ensemble des compartiments représentés était trop conséquent pour être affiché en totalité dans la fenêtre (déjà dimensionnée à la taille de l'écran). Encore une fois, ce problème pouvait ne pas être ressenti chez les utilisateurs disposant d'un écran suffisamment large ou d'une installation à double écran. Pour corriger ce bug, une *scrollbar* (barre de défilement) est logiquement établie à ce niveau-là afin de faire défiler les paramètres sans avoir à retoucher la taille des fenêtres. Pour ce faire, ces compartiments déjà implémentés sous la forme de `frames` l'an dernier, sont cette fois-ci placés sur un `canvas`, et non à même la fenêtre, permettant ainsi de construire un domaine « scrollable » en python. Pour ce faire,

la fonction `displayCompParam()` du fichier `GUI.py` a complètement été modifiée avec notamment des méthodes `tkinter`, `Scrollbar` et `Canvas`.

L'ultime amélioration pour cette interface, concernait les menus déroulant du paramètre `Add injection` lors de la sélection d'une injection et de son type. En effet, ces derniers étaient semi-fonctionnels puisque les sélections s'établissaient bien au sein de la simulation mais n'étaient pas visibles pour l'utilisateur dans les champs des volets fermés (post-sélection). Ce bug était lié au type des fenêtres `tkinter` utilisées, au sein desquelles se trouvaient les menus déroulants, qui ont ainsi été remplacées par des fenêtres dites primaires. Bien qu'elles aient les mêmes caractéristiques qu'une fenêtre classique, les fenêtres primaires possèdent une existence indépendante pour le gestionnaire de fenêtres du système d'exploitation. Cette dernière propriété a alors permis de contourner efficacement le problème. Ainsi, la nature des fenêtres de la fonction `getInjectionCreationWindow` du fichier `GUI.py` a été modifiée et est passée de fenêtres `Tk()` à des fenêtres `Toplevel`.

Par la suite, les modifications faites sur l'interface graphique ne sont pas des corrections d'affichage à proprement parler mais plutôt des améliorations visuelles afin de le rendre le plus agréable possible.

2.1.2 Modèles et simulations

Le programme doit permettre de charger les deux nouveaux modèles (rongeur 3 populations et rongeur 6 populations) ainsi que le modèle initial (humain 3 populations). Les fichiers textes contenant ces modèles seront donc formalisés de la même manière. La figure 2.1 montre le comparatif de ces fichiers.

Dès qu'un modèle est importé par `main.py`, ses paramètres sont stockés dans la classe `Network`. Le bouton `Display Compartments Parameters` de l'interface permet l'affichage d'un canevas dans la fenêtre contenant tous les paramètres des populations neuronales, de l'homéostasie et des neurotransmetteurs à injecter. Sur la base de l'affichage, tous les paramètres de cette fenêtre peuvent être modifiés par l'utilisateur et enregistrés dans un nouveau fichier `.txt`. De plus, le perfectionnement du programme permet à présent à l'utilisateur d'ajouter de nouveaux compartiments (`population neuronale` ou `homéostasie`) dans le `Network` et d'établir les nouvelles connexions avec le bouton `Add Object to Network`. Les nouveaux compartiments et nouvelles connexions s'affichent alors dans le panneau des paramètres.

Après confirmation des paramètres par l'utilisateur, le programme calculera les fréquences des décharges de chaque population neuronale par unité de temps selon les modèles mathématiques de régulation.

Pour intégrer le modèle à 6 population, il a été nécessaire de rajouter la régulation circadienne *CIRC(t)* dans `SleepRegulationOOP.py` lorsque la population cible est SCN comme le montre la fonction du listing 2.1 :

Listing 2.1 – Nouvelle fonction getI

```
def getI(self,dt,N):
        result = 0
        for c in self.connections:
            if c.type == "NP-NP":
                result += c.getConnectVal(N)
        if self.name == "SCN":
            result += np.sin((2*np.pi*N*dt)/(24*3600))
        return result
```

modèle de 3 populations [humain]	modèle de 3 populations [rongeur]	modèle de 6 populations [rongeur]	modèle de 6 populations [lésion]
* population = wake promoting = WAKE F = 6.0 F_max = 6.5 alpha = 0.5 beta = -0.4 tau_pop = 1500E3 g_NT_pop_list = -1.68 1.0 pop_list = NREM REM neurotransmitter = noradrenaline C = 0.9 gamma = 5.0 tau_NT = 25E3 *	* population = wake promoting = WAKE F = 0.5 F_max = 6.5 alpha = 0.5 beta = -0.23 tau_pop = 25000 g_NT_pop_list = -2.0 1.2 pop_list = NREM REM neurotransmitter = noradrenaline C = 0.9 gamma = 5.0 tau_NT = 25E3 *	* population = LC ... *	* population = LC promoting = WAKE g_NT_pop_list = 3.5 3.5 -1.5 -1.5 **0** pop_list = R WR LC VLPO **SCN** F = 6.0 C = 0.9 F_max = 6.5 beta = -1.85 alpha = 0.75 tau_pop = 25000 neurotransmitter = noradrenaline_LC gamma = 5.0 tau_NT = 25000 *
* population = NREM ... *	* population = NREM ... *	* population = DR ... *	* population = DR *
* population = REM ... *	* population = REM ... *	* population = VLPO ... *	* population = VLPO *
+ cycle = homeostatic h = 0.5 H_max = 1.0 tau_hw = 34830E3 tau_hs = 30600E3 pop_list = wake g_NT_pop_list = 0.0 theta = 2 +	+ cycle = homeostatic h = 0.5 H_max = 1.0 tau_hw = 600000 tau_hs = 380000 pop_list = wake g_NT_pop_list = 0.0 theta = 2 +	* population = WR ... *	* population = WR *
#	#	* population = R ... *	* population = R *
t = 0 T = 86400 res = 50 mean = 0 std = 0.001 #	t = 0 T = 3600 res = 50 mean = 0 std = 0.001 #	* population = SCN ... *	* population = SCN *
& neurotransmitter = noradrenaline agoniste = antagoniste = imin = imax = &	& neurotransmitter = noradrenaline agoniste = antagoniste = imin = imax = &	+ cycle = homeostatic ... +	+ cycle = homeostatic ... +
& neurotransmitter = acetylcholin ... &	& neurotransmitter = acetylcholin ... &	# ... #	#
& neurotransmitter = GABA ... &	& neurotransmitter = GABA ... &	& neurotransmitter = GABA_VLPO ... &	... #
		& neurotransmitter = GABA_SCN ... &	& neurotransmitter = GABA_VLPO &
		& neurotransmitter = acetylcholin_WR ... &	& neurotransmitter = GABA_SCN &
		& neurotransmitter = acetylcholin_R ... &	& neurotransmitter = acetylcholin_WR &
		& neurotransmitter = serotonin_DR ... &	& neurotransmitter = acetylcholin_R &
		& neurotransmitter = noradrenaline_LC ... &	& neurotransmitter = serotonin_DR &
			& neurotransmitter = noradrenaline_LC &

FIGURE 2.1 – **Les formats des différents modèles.** Les blocs gris correspondent à tous les paramètres d'une population neuronale. Ils sont délimités par le symbole $*$. Les blocs bleus correspondent à tous les paramètres du processus d'homéostasie. Ils sont délimités par le symbole $+$. Les blocs jaunes définissent les paramètres de la simulation, tels que la durée totale et l'intervalle de temps entre chaque résultat enregistré. Ils sont délimités par un symbole #. Les blocs verts permettent de transmettre les paramètres pour effectuer une micro-injection. Ils sont délimités par un symbole &. Le modèle lésion est le même que e modèle 6 populations mais les poids du SCN sont initialisés à 0 pour couper les connexion (en rouge dans la quatrième colonne).

Cette fonction du listing 2.1 permet de récupérer les produits entre les poids synaptiques et les concentrations en neurotransmetteurs puis les additionne. Elle teste également le nom de la population en cours d'itération car si cette population est SCN, il faut ajouter la régulation circadienne au résultat de l'addition. Cette régulation est propre au modèle 6 populations et a dû être ajoutée au programme initial pour faire fonctionner le nouveau modèle.

2.1.3 Lésions et injections

Le modèle de lésion des 6 populations est utilisé pour évaluer l'effet de la population SCN sur les autres populations neuronales en lésant toutes les connexions entre SCN et LC, DR, VLPO, R. Pour ces 4 populations, le poids synaptique correspondant à SCN est remplacée par 0. Cette opération peut également être effectuée dans le panneau des paramètres grâce à la correction d'un bug pré-existant. Dans le fichier texte de ce modèle, cela se situe au niveau des lignes `g_NT_pop_list` de chaque population neuronale cible concernée.

La fonction `read_parameters()` du fichier `manage_parameters.py` se charge de trier les paramètres des neurotransmetteurs liés aux injections (blocs verts dans la figure 2.1) issus des fichiers modèles. Une classe `ParaInjection` a été ajoutée dans le fichier `SleepRegulationOOP.py` et fait le lien avec la classe `Network` (du même fichier). Cela permet à la classe `NetworkGUI` du fichier `GUI.py` d'appeler ces paramètres. Le dictionnaire de `Network` ajoute les éléments dont les noms des clés sont les noms des neurotransmetteurs injectés. Cela affectera d'autres fonctions qui doivent parcourir tous les éléments de l'ancien dictionnaire. Par conséquent, une liste `self.nlist` (listing 2.2) est initialisée dans la classe `Network`. En ajoutant une structure conditionnelle (listing 2.3) dans ces fonctions, le programme peut s'exécuter comme prévu.

Listing 2.2 – Liste des neurotransmetteurs impliqués dans les modèles

```
self.nlist = ['GABA_VLPO', 'GABA_SCN', 'acetylcholin_WR', 'acetylcholin_R', '
    noradrenaline_LC', 'serotonin_DR', 'noradrenaline', 'acetylcholin', 'GABA']
```

Listing 2.3 – Structure conditionnelle ajoutée

```
for c in self.compartments.keys():
            if c not in self.nlist:
```

Le fichier `GUI.py` propose une fenêtre secondaire comportant un menu de sélection afin d'effectuer les commandes d'injection par l'exécution de la fonction `getInjectionCreationWindow (self)`. Une fois que l'utilisateur a sélectionné un type de neurotransmetteur et la population cible, la fonction `getName(name)` collecte la réponse et affiche tous les paramètres de ce neurotransmetteur dans une fenêtre de troisième ordre. La nature des neurotransmetteurs (agoniste ou antagoniste) doit être sélectionnée. D'autre part, les paramètres peuvent aussi être modifiés directement dans le tableau.

La fonction `self.addInjection()` du fichier `SleepRegulationOOP.py` sert à modifier le type de connexion entre la population source et la population cible à `NP-MIE-NP` ou `NP-MII-NP` puis utilise les paramètres collectés pour créer des instances basées sur la classe `Injection`. Selon les équations mathématiques de l'article [Diniz Behn et Booth, 2010], la concentration de neurotransmetteur injecté dans le réseau de neurones artificiel est augmenté ou diminué proportionnellement par unité de temps. Par conséquent, les résultats des fréquences des décharges de chaque population neuronale par unité de temps calculés par le programme montreront l'influence de le simulation des micro-injections.

Deux classes indépendantes (`Classe ParaInjection` et `Classe Injection`) garantissent qu'après avoir modifié les données dans l'interface, le modèle d'injection peut être exécuté en fonction des nouvelles données. De plus, cela élimine la duplication des données et des fonctions créées pour les types agoniste et antagoniste.

2.1.4 Visualisation graphique

Lors de l'intégration du modèle à six population, il s'est avéré nécessaire d'améliorer la représentation graphique des courbes car ces dernières avaient parfois des couleurs difficilement distinguable.

Dans le fichier `graphic.py`, il y a des couleurs attribuées aux populations faisant partie du modèle à trois populations puis une liste d'autres couleurs est créée. Lorsque des populations différentes sont

ajoutées, une couleur de cette liste est attribuée aléatoirement. Cela est, en effet, pratique pour que l'utilisateur ajoute de nombreuses autres populations lui même. Cependant, il a paru judicieux de fixer les couleurs pour les populations des modèles créés dans ce projet de sorte à fournir des graphiques de qualité. Par conséquent, des couleurs ont été attribuées en suivant la même méthode que pour le modèle à trois populations dans la fonction `createGraph()`. Il s'agit en fait de définir dans un dictionnaire le nom d'une population en tant que clé et la couleur sous forme de chaîne de caractères en tant que valeur associée.

Afin d'améliorer la représentation graphique et le confort de visualisation, il a été nécessaire de redimensionner la fenêtre contenant les graphes. Pour ce faire, l'argument `figsize` a été ajouté à la ligne de code permettant de générer les graphes. Ensuite, les `subplots` ont été redimensionnés aussi de sorte à laisser un espace suffisant entre chaque graphe pour ajouter des titres qui étaient absents. Un espace a également été ajouté à droite des graphes afin de pouvoir exclure les légendes des graphes. En effet, le programme initial permettait l'affichage des légendes directement sur les courbes ce qui pouvait gêner la visualisation. Par conséquent, les légendes ont été fixées à l'extérieur des graphes pour pallier ce problèmes. L'utilisation de `chartBox` (création d'une variable, comme une sorte de boite qui contient les informations du positionnement du graphe) a été privilégiée. Cette méthode permet de récupérer la position du graphe de modifier sa position et d'en modifier la dimension si nécessaire. A l'aide de cela, la légende est placée à sur le côté droit de la `chartBox`. Le placement est calculé par rapport à la hauteur et la largeur de la `chartBox`.

Toujours dans la fonction `createGraph()`, il a été nécessaire d'ajouter une structure conditionnelle pour la création de l'axe de temps puisque l'échelle n'est pas la même selon qu'il s'agisse du modèle humain ou rongeur. En effet, pour l'un c'est en heures et pour l'autre en minutes. La structure conditionnelle teste si la dernière valeur de temps stockée dans le dictionnaire `data['time']` est inférieure ou égale à 3600 secondes (une heure). Si c'est le cas, alors l'axe affichera des minutes. Si ce n'est pas le cas, alors l'axe s'affichera en heures. Le test est effectué également pour adapté le label de l'axe.

Cet axe de temps a de nouveau été amélioré en ajoutant les valeurs extrêmes qui étaient absentes et en améliorant les étiquettes pour le modèle humain. Les pas de temps sont stockés dans des listes mais la dernière valeur ne correspondait pas forcément à la valeur extrême. Par exemple, pour une simulation de 3600 secondes, la dernière valeur retenue peut être 3598 secondes (cela dépend du nombre d'itération lors de la simulation). Par conséquent, la dernière valeur n'apparaissait pas sur le graphe puisque les étiquettes se positionnent à intervalles réguliers. Cela a été pallié en ajoutant la valeur extrême directement à la suite de la liste des pas de temps. Ainsi lorsque les étiquettes sont créées, la dernière valeur est bien présente. Les étiquettes pour les modèles rongeurs sont alors réparties toutes les quinze minutes entre 0 et 60 minutes. Pour le modèle humain, le paramétrage a été passé d'un intervalle de cinq heures à un intervalle de deux heures. Une étiquette est donc placée sur l'axe toutes les deux heures entre 0 et 24 heures.

Dans la fonction `createMeanGraphs()`, qui calcule les courbes moyennes de plusieurs fichiers de résultats, le calcul des décharges neuronales et des concentrations a été réalisé de manière à être transférable quel que soit le modèle chargé. Cependant, ce n'est pas le cas pour le calcul de l'hypnogramme moyen car ce dernier présente utilise les clés du dictionnaire. Or, le nom des clés diffère entre le modèle à trois populations et le modèle à six populations. Par conséquent, une structure conditionnelle a dû être ajoutée afin de vérifier l'existence de la dite clé.

Cette partie a également été optimisée de sorte que le nom des clés utiles à cet endroit soient stockées dans des variables. Ainsi, la structure conditionnelle affecte les bons noms de clés dans des variables (`key_wake_C, key_REM_C`). Par conséquent, la structure itérative, permettant la création de l'hypnogramme moyen, manipule les variables et non plus les chaînes de caractères directement.

D'autre part, si les clés testées ne correspondent pas aux clés attendues, alors le programme quitte la fonction en cours et un message est retourné à l'utilisateur sans arrêter le programme.

2.1.5 Statistiques

Le fichier `Stats.r` permet de faire les tests et calculs statistiques sur les fichiers `.csv` contenant les résultats des simulations. Cependant, ce fichier ne fonctionnait pas correctement car certaines commandes sont maintenant obsolètes.

Il a été nécessaire de modifier la commande chargée de lire les fichiers `.csv` (`read.csv`). En la remplaçant par `read.table`, le script peut lire les fichiers de résultats et stocker ces derniers dans des variables.

D'autre part, lors de la réalisation des tests lésionnels, des données pouvaient être manquantes. Cependant, le test chargé de comparer les variances (`ANOVA`) ne prend pas en charge les valeurs manquantes. Il était nécessaires de les transformer en `NA`. Or, cela n'est plus possible à présent. La valeur '0' a donc été attribuée aux valeurs manquantes par la commande `boutDurDF[boutDurDF == "NULL"] <- 0` afin de pouvoir pratiquer le test.

2.2 Structure du programme et Évolution

2.2.1 Architecture

La figure 2.2 présente l'architecture du modèle dans sa version 2020, implémenté en Python et R. Les noms de fichiers ainsi que l'architecture générale du programme n'ont globalement pas été modifiés. Cependant, il a été nécessaire modifier voir ré-implémenter ou ajouter certaines fonctions.

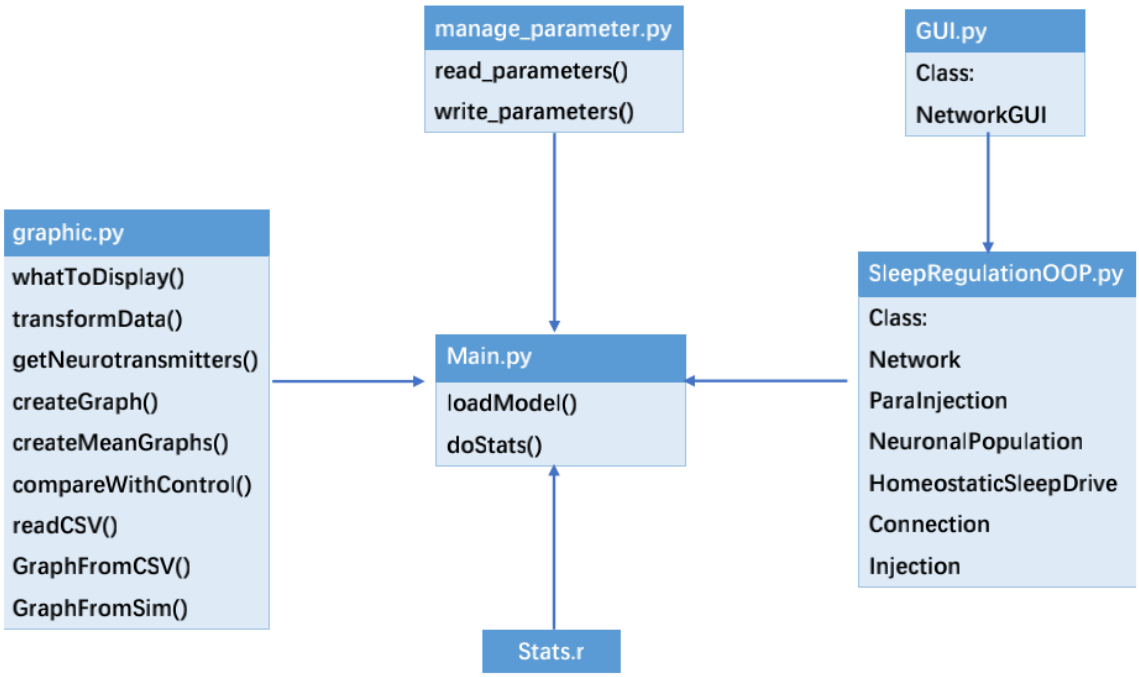


FIGURE 2.2 – **L'architecture du programme.** Ce diagramme représente chaque fichier constituant le programme développé dans ce projet. Les flèches montrent les interactions entre ces fichiers. Chacun de ces derniers présente la liste des fonctions et/ou des classes qui y sont implémentées.

2.2.2 Diagramme des classes

La figure 2.3 montre les classes nécessaires au fonctionnement du programme réalisé au cours de ce projet.

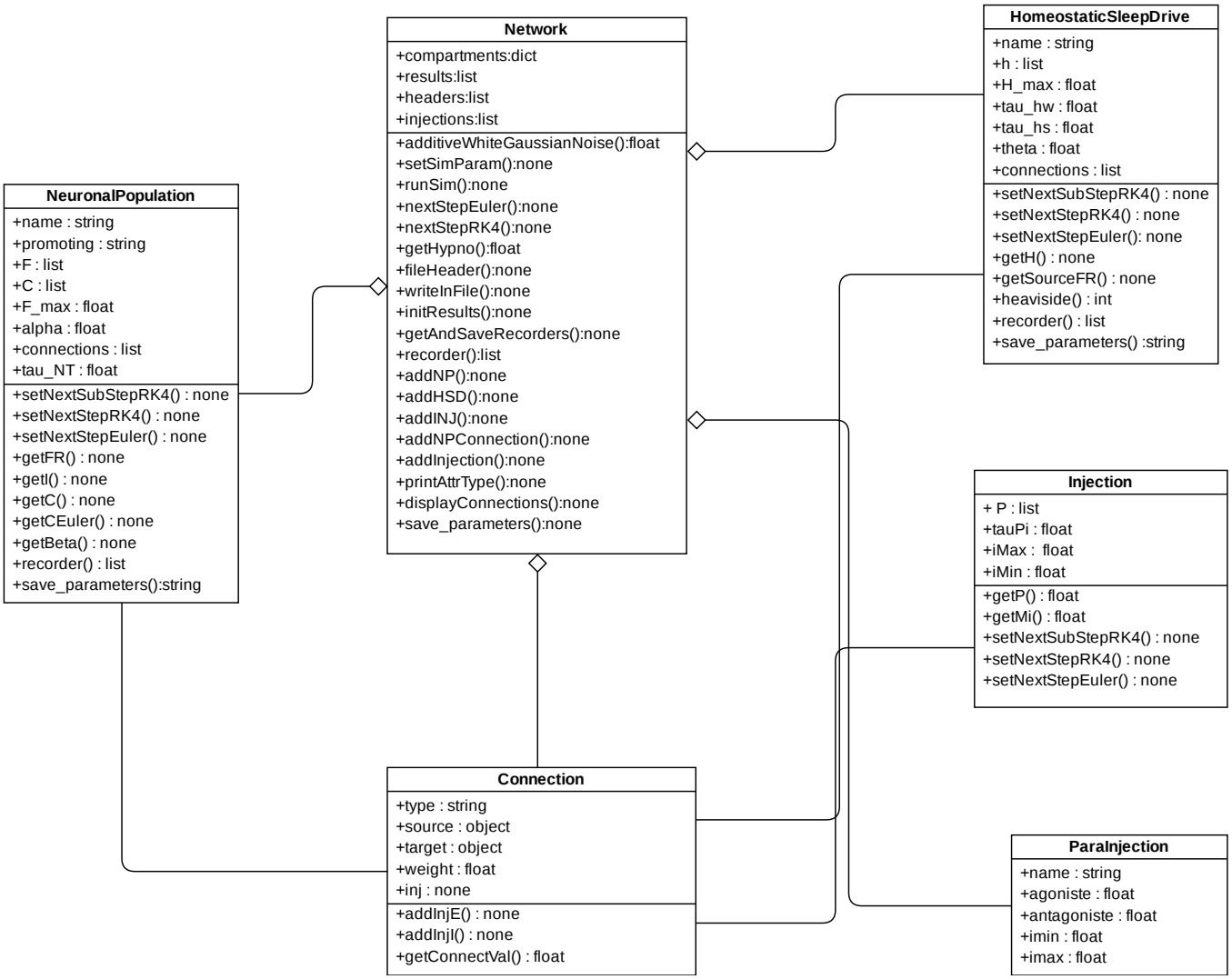


FIGURE 2.3 – **Diagramme des classes du programme de 2020.** Ce diagramme représente les 6 classes du programme issu de ce projet. La classe `Network` est aux centres des interactions. Les flèches avec un bout carré signifient que les 2 classes ont une relation d'agrégation. Le liaisons sans flèche représentent une association.

2.2.3 Tableau d'améliorations

La figure 2.4 présente succinctement les améliorations apportées lors de ce projet sur la base du programme des étudiants de Bordeaux de 2019 [Darnige *et al.*, 2019].

Fichiers originaux du modèle 2019	Améliorations et apports du programme 2020
`Main.py`	Modification de la taille des fenêtres Ajout des données liées aux injections dans `Network`
`GUI.py`	Modification de la fonction `displayCompParam()` : création d'un canevas et ajout d'une barre de défilement Changement du type de fenêtre Correction des bugs : - Permettre la modification et la sauvegarde des paramètres depuis l'interface - Permettre l'ajout d'un compartiment (population neuronale ou homéostasie) dans `Network` Modification et correction des fenêtres permettant les injections Ajout de fonctionnalités dans les injections (chargement automatique des paramètres et optimisation)
`graphic.py`	Modification des couleurs des courbes Modification de la taille de la fenêtre Modification de l'espacement des *subplots* Ajout des titres des *subplots* Amélioration de l'affichage des légendes Améliorations de l'axe du temps pour tous les modèles Ajout des étiquettes de valeurs extrêmes sur l'axe du temps.
`manage_parameters.py`	Modification des fonctions pour permettre aux paramètres des injections d'être reconnus lors du chargement et de la sauvegarde du modèle
`SleepRegulationOOP.py`	Ajout une nouvelle classe `ParaInjection` Modification de toutes les fonctions pour adapter `Netwok` avec `ParaInjection`
`Stats.r`	Mise à jour des commandes obsolètes : - `read.table` à la place de `read.csv` - Transformation des données manquantes en valeurs nulles dans le test ANOVA.
Modèles existants : - 3 populations humain(fonctionnel), - 5 populations rongeur (non fonctionnel)	Modification du modèle humain Ajout du modèle rongeur à 3 populations, à 6 populations avec et sans lésion du SCN

FIGURE 2.4 – **Tableau des améliorations apportées en 2020.** Ce tableau est une synthèse des améliorations apportées au programme de 2019 [Darnige *et al.*, 2019] lors de ce projet.

2.3 Résultats et Discussion

2.3.1 Utilisation du programme

Dans cette section, seuls les ajouts et modifications du programme apportés par le projet de cette année seront présentés. L'utilisation de l'application est détaillée dans le manuel d'utilisation fournit au client.

La figure 2.5 met en avant de tous les modèles rongeur disponibles et prêts à l'emploi. Ils sont accompagnés des modèles humain crées en 2019, non dénaturés et toujours aussi fonctionnels.

La figure 2.6 met en évidence la `scrollbar` qui permet de faire défiler les paramètres vers la droite pour ainsi faire apparaître des populations encore invisibles à l'écran comme `SCN`.

La figure 2.7 montre que le menu déroulant, une fois fermé, affiche désormais la sélection au sein d'un champ.

La figure 2.8 montre d'une part un modèle fonctionnel avec des courbes lisibles ainsi qu'une échelle de temps adaptée aux modèles rongeurs (60 minutes) ; d'autre part elle met en évidence les modifications de présentation apportées avec les légendes (entourés en bleu) cette fois ci placées en dehors de chaque graphiques, avec des titres associés à chacun d'eux (entourés en rouge). Cette conformation est valable pour tous les modèles disponibles et l'échelle de temps s'adapte en fonction du le type de modèle. Ainsi, les graphiques des simulations sur les humains sont toujours exploitables sur 24h.

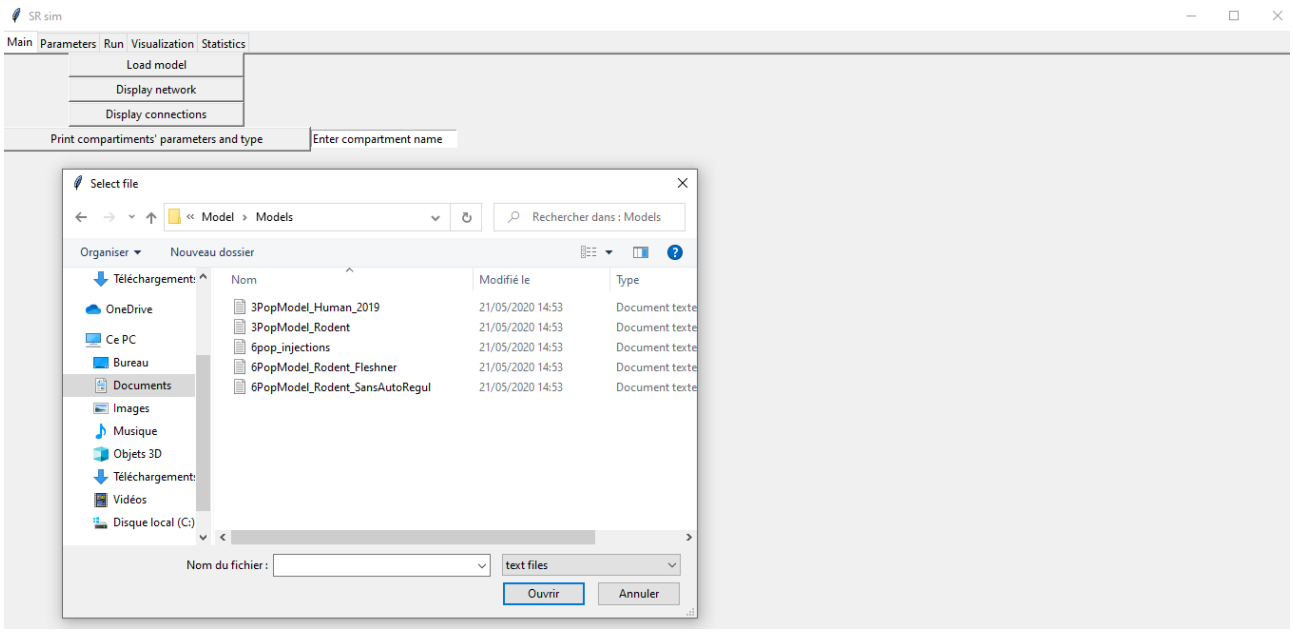


FIGURE 2.5 – **Affichage lors du chargement d'un modèle.** Affichage de la fenêtre de sélection d'un modèle générée par le bouton `Load model` (système d'exploitation : Windows).

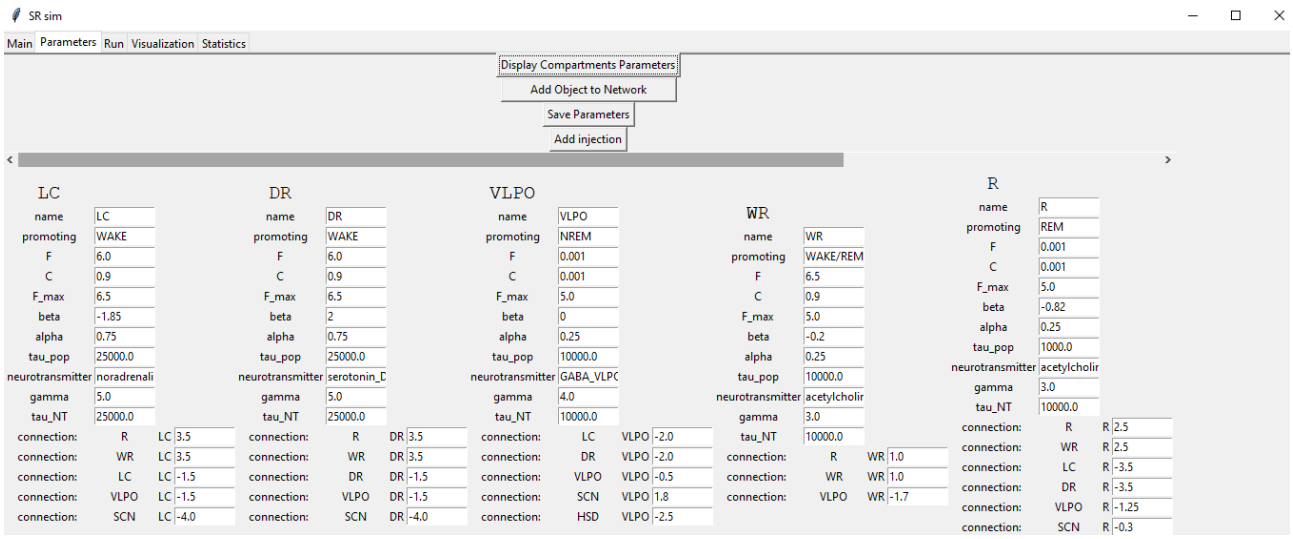


FIGURE 2.6 – **Scrollbar.** Affichage des paramètres attribués à chaque populations de neurones pour le modèle à 6 populations.

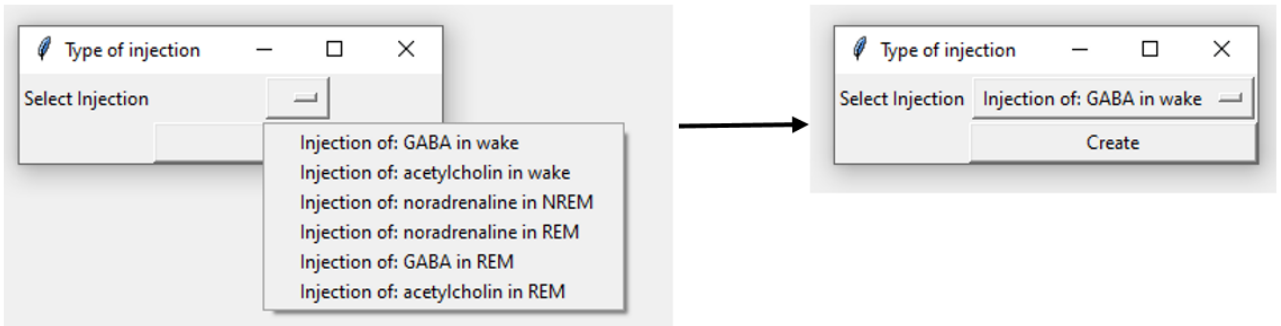


FIGURE 2.7 – **Menu des injections**. Affichage de la fenêtre de sélection du type d'injection après avoir actionné le bouton `Add injection`. A gauche de la flèche, la fenêtre propose un menu déroulant avec tous les types d'injection. A sa droite, il s'agit de la même fenêtre après sélection.

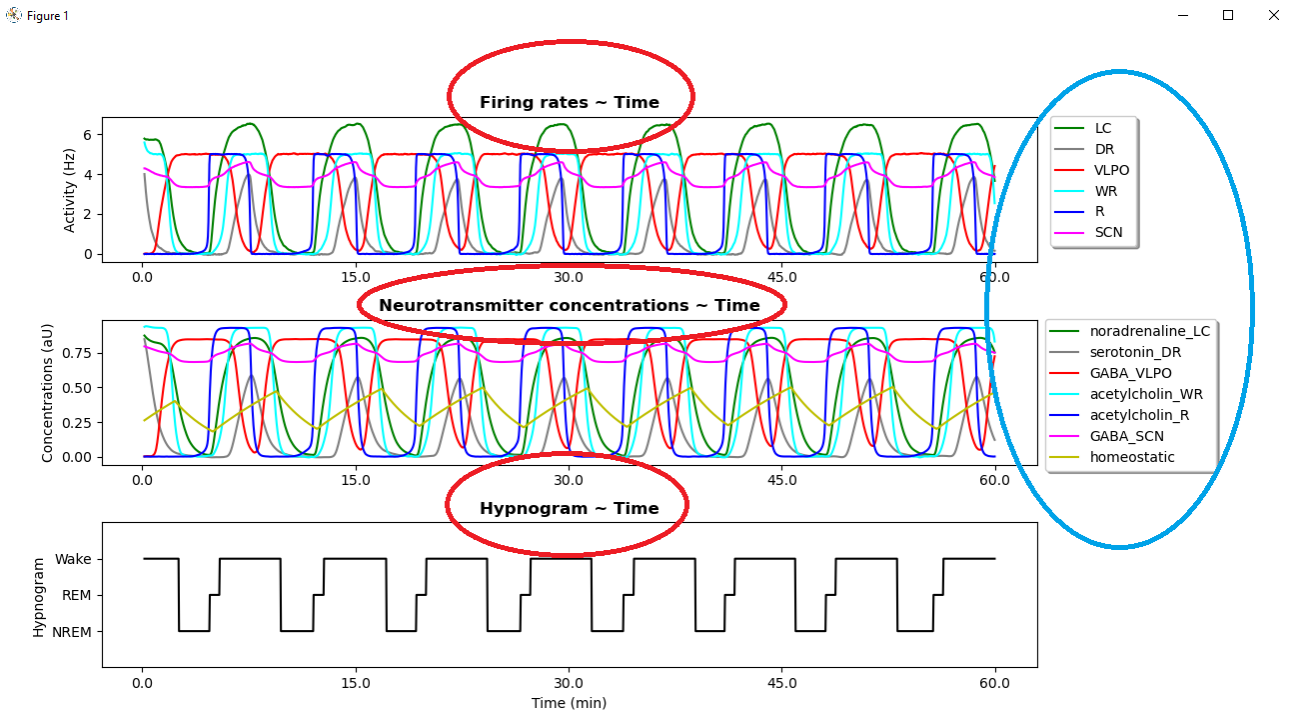


FIGURE 2.8 – **Visualisation graphique.** Fenêtre de la visualisation des résultats issus d'une simulation sur le modèle rongeur à 6 population.

2.3.2 Conformité bibliographique

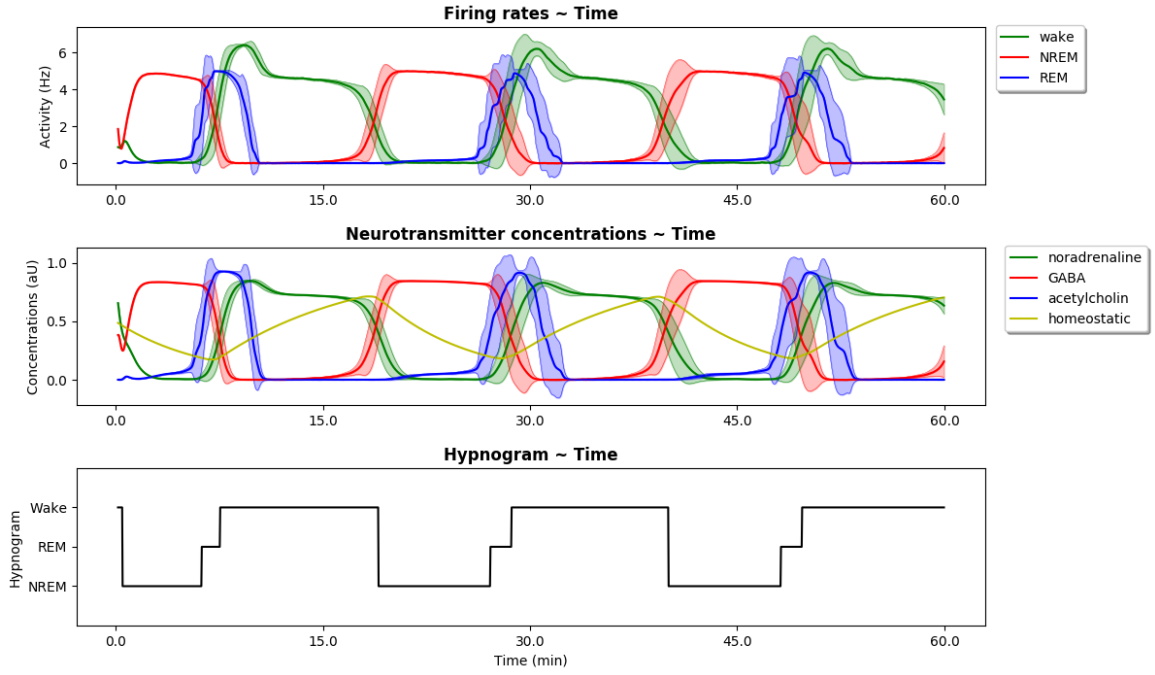


FIGURE 2.9 – **Graphe moyen du modèle rongeur à 3 populations de neurones (n=10 simulations). Firing rates-Time** représente la fréquence de décharge de chaque population en fonction du temps. **Neurotransmitters concentration-Time** est la représentation de l'évolution des concentration en neurotransmetteurs en fonction du temps. **Hypnogram-Time** modélise l'état dans lequel se trouve le réseau sur une heure (hypnogramme).

La figure 2.9 témoigne de la conformité de ce modèle vis à vis des articles de référence [Fleshner *et al.*, 2011, Schellenberger Costa *et al.*, 2016, Diniz Behn et Booth, 2010, Diniz Behn et Booth,

2012]. En effet, ces graphiques sont le résultat de 10 simulations exécutées sous le modèle rongeur `3PopModel-Rodent`. L'allure des courbes est conforme à celles retrouvées dans l'article scientifique utilisé pour ce modèle [Diniz Behn et Booth, 2012], avec un hypnogramme associé cohérent. De plus, les écarts-types restent relativement proches des courbes moyennes, ce qui certifie la bonne reproductibilité des résultats.

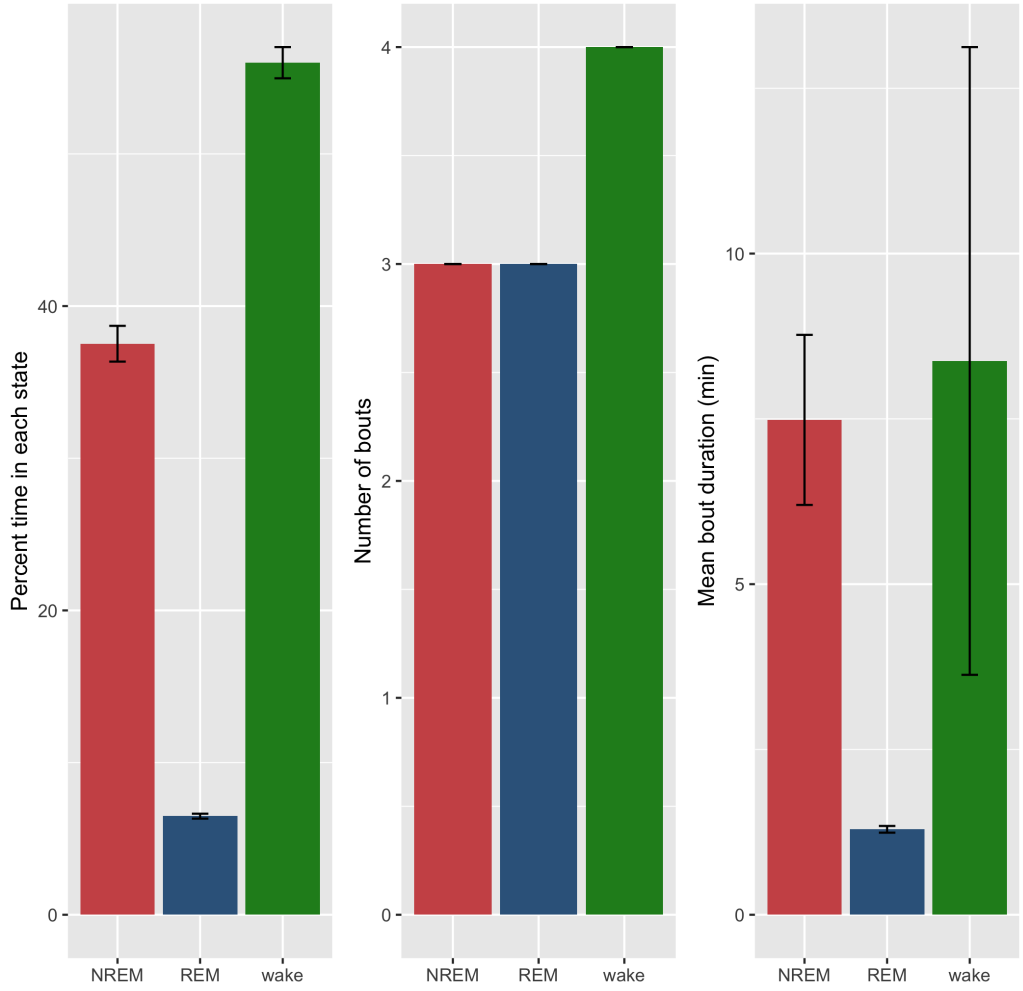


FIGURE 2.10 – **Histogrammes résumant les différents états du sommeil du modèle à 3 populations de neurones.** En fonction des 3 états NREM, REM et wake, ces histogrammes représentent respectivement : le pourcentage de temps pour chaque état, le nombre de fois où ils sont effectifs et la durée moyenne de chacun d'eux. Les segments de couleur noir modélisent les écarts-types.

La figure 2.10 reprend l'hypnogramme de la figure 2.9. Il permet alors de décomposer sous la forme de d'histogrammes les différentes phases du sommeil chez les rongeurs, pendant 1 heure, afin d'en tirer une conformité biologique. Ces résultats, pouvant largement varier d'un individu à l'autre, restent conformes à l'article *"A Fast-Slow Analysis of the Dynamics of REM Sleep"* [Diniz Behn et Booth, 2012]. Il est à noter que la durée moyenne du wake et du NREM sur le dernier graphique peut varier considérablement, comme le témoignent les écart-types. En effet, cela dépend de la phase au sein de laquelle commence la simulation. Ici, elle peut débuter soit en phase de NREM soit en période d'éveil.

Les différentes courbes de la figure 2.11 sont conformes aux résultats attendus. En effet, les graphiques **A.** et **B.**, correspondent à ceux retrouvé dans l'article **"Circadian regulation of sleep–wake behaviour in nocturnal rats requires multiple signals from suprachiasmatic nucleus"** [Fleshner *et al.*, 2011]. En ce qui concerne les graphiques moyens des injections, ils sont tous aussi comparables à ceux décrit dans l'étude édifiée par Diniz Behn et son équipe en 2010 [Diniz Behn et Booth, 2010].

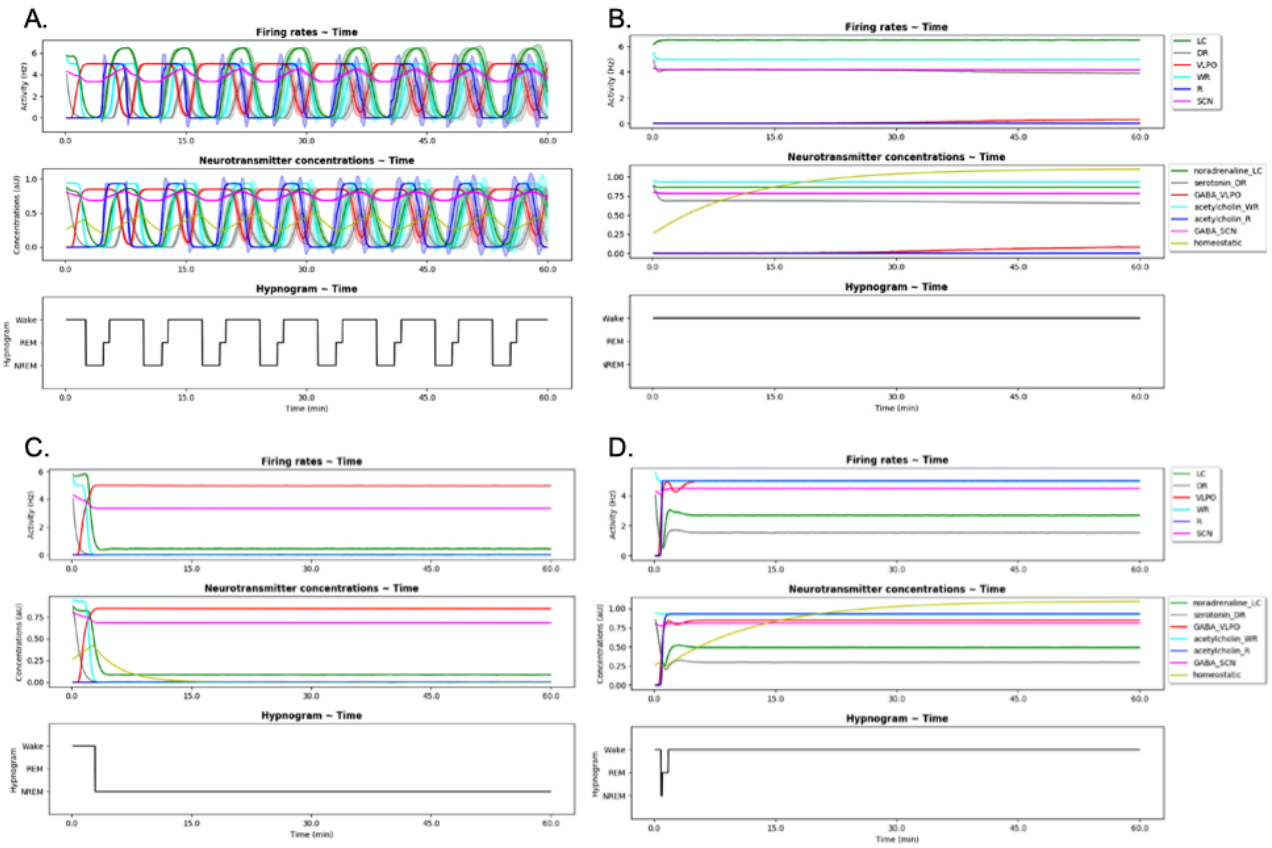


FIGURE 2.11 – **Graphiques moyens des différents modèles rongeurs basés sur la représentation à 6 populations de neurones. A.** Graphique moyen du modèle rongeur à 6 population de neurones à l'état physiologique. **B.** Graphique moyen du modèle à 6 populations lésé au niveau du `SCN`. **C.**Graphique moyen du modèle à 6 populations avec une injection à effet antagoniste(GABA). **D.** Graphique moyen du modèle à 6 populations avec une injection à effet agoniste(ACh).

Comme pour la figure 2.10, les graphiques de la figure 2.12 reprennent l'hypnogramme de chaque modalités expérimentales. Ils s'interprètent donc de la même manière. Enfin, ils sont eux aussi conformes aux articles à partir desquels leurs modèles respectifs sont fondés. [Fleshner *et al.*, 2011][Diniz Behn et Booth, 2010]

2.4 Axes d'amélioration

Le travail réalisé a permis d'améliorer considérablement les fonctionnalités et l'efficacité du programme. En revanche, il est possible de l'optimiser davantage.

Tout d'abord, afin de permettre une meilleure interprétation graphique des résultats, il serait intéressant d'afficher automatiquement le graphique moyen de contrôle en parallèle des graphes issus de la simulation. Cet affichage permettrait, d'une part, de voir plus facilement qu'une simulation est conforme aux représentations attendues et d'autre part, de comparer directement l'état physiologique aux résultats post-lésionnels ou post-injections de neurotransmetteurs .

D'autre part, depuis que l'interface graphique est dimensionnée à la taille de l'écran, il n'est plus possible pour l'utilisateur de voir le terminal en parallèle de l'interface (à moins d'avoir un double écran ou de redimensionner manuellement les fenêtres). Par conséquent, les messages destinés à l'utilisateur, générés par la fonction `print()` dans la console, ne sont plus visibles. Ces derniers indiquent notamment la bonne exécution du chargement du modèle, l'établissement des connexions, l'avancement du chargement d'une simulation, etc. Il serait donc plus confortable pour l'utilisateur d'afficher

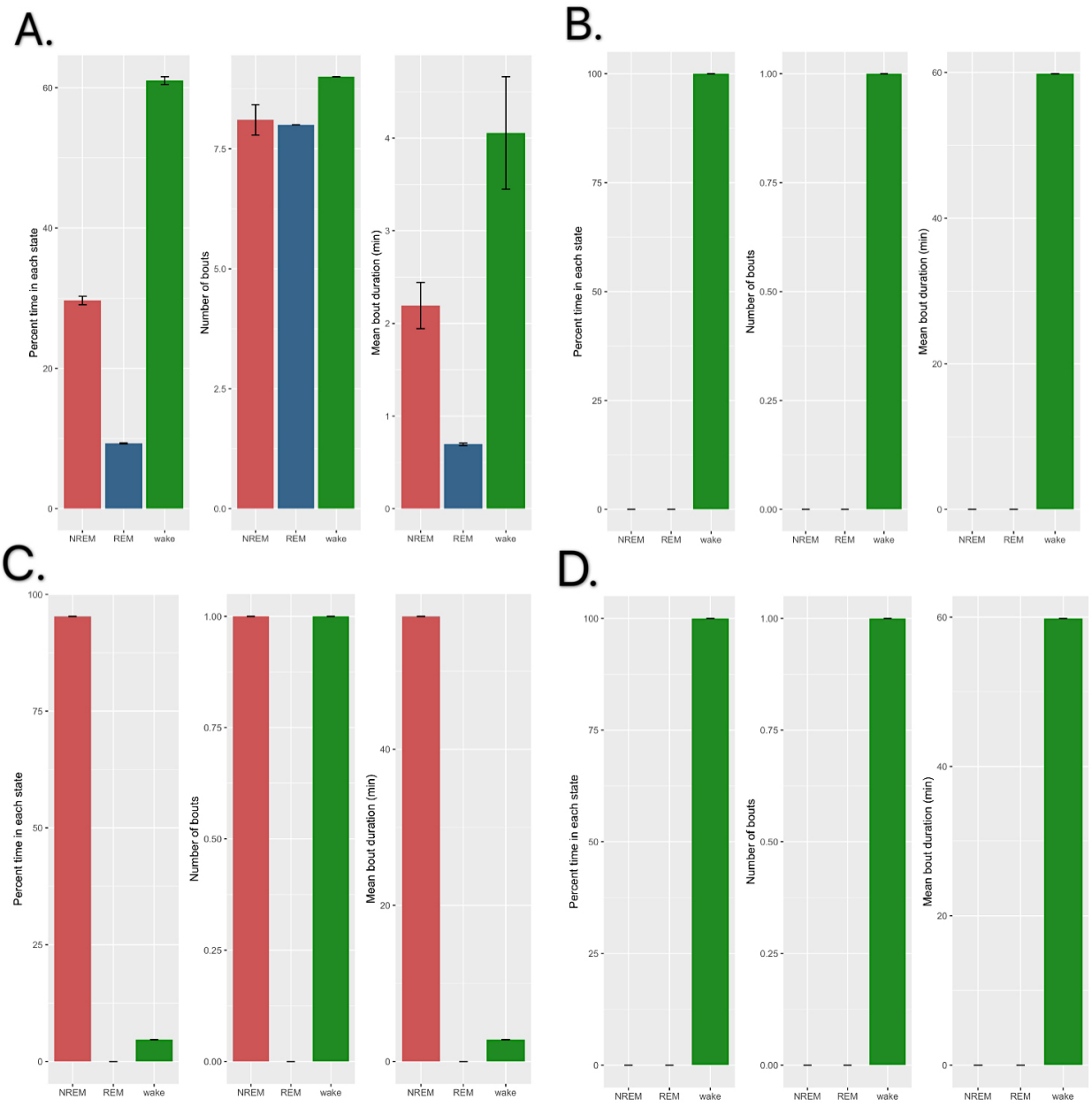


FIGURE 2.12 – **Histogrammes représentant les différentes phases du sommeil basées sur modèle rongeur à 6 populations de neurones et ses variantes. A.** Histogramme du modèle rongeurs à 6 populations de neurones à l'état physiologique. **B.** Histogramme du modèle rongeurs à 6 populations de neurones, lésé au niveau du `SCN`. **C.** Histogramme du modèle rongeurs à 6 populations de neurones avec une injection à effet antagoniste (GABA). **D.** Histogramme du modèle rongeurs à 6 populations de neurones avec une injection à effet agoniste (ACh). Les segments de couleur noir modélisent les écarts-types.

ces informations directement sur l'interface graphique.

Enfin, il serait opportun de modifier la présentation de l'interface afin de la rendre esthétiquement plus agréable.

Conclusion

Pour conclure et en reprenant les objectifs mentionnés précédemment, de la création des divers modèles à la modification de l'interface graphique pour les accueillir, ce projet répond aux attentes du client.

En effet, l'ensemble des modèles humain et rongeur sont fonctionnels. Les simulations de lésions et d'injections sur chacun d'eux sont opérationnelles. Les résultats et visualisations graphiques sont clairs et permettent une bonne interprétation biologique. De plus, les différents paramètres sont correctement visibles au niveau de l'interface et peuvent désormais être sauvegardés par l'utilisateur. Le manuel utilisateur (*User Guide*) a également été mis à jour.

Ces résultats ont été obtenus grâce au travail d'équipe et à la supervision conjointe du Dr Héricé (initiatrice et cliente de ce projet) et du Dr Beurton-Aimar (responsable de l'enseignement dans lequel s'inclut ce projet). Ce travail a fait l'objet d'une première collaboration sur un projet de programmation de moyenne ampleur. Pour ce faire, de nombreux outils tels qu'un gestionnaire de tâches collaboratif et des logiciels de visio-conférence et de messagerie instantanée ont été employés afin d'organiser et de répartir les tâches. Le dépôt et gestionnaire de version GitHub a été nécessaire à la programmation collaborative. La plateforme Overleaf a, quant à elle, été employée pour la rédaction des différents travaux écrits tel que ce rapport.

La rigueur et l'entraide ont été des atouts déterminant pour la réussite de ce projet sans oublier le soutien et l'expertise des professionnels encadrants. La communication a été le maître mot tout au long de la réalisation de ce projet et a permis de le mener à bien. Cette première expérience a rendu plus concret le rôle et le travail du bioinformaticien en équipe de recherche. De plus, il a été nécessaire de s'inclure dans un projet déjà initié par une autre équipe ce qui est conforme à la réalité professionnelle. Cela a permis d'asseoir les compétences de chacun et de mettre l'épreuve la capacité d'adaptation capitale dans ce métier.

Bien que le travail réalisé soit abouti et réponde aux attentes fixées par le client, l'application n'est pas au bout de son domaine d'exploitation et serait très probablement en capacité d'effectuer d'autres types de simulations. Les nouveaux modèles pourraient alors compter plus de populations neuronales afin de se rapprocher au maximum de la réalité biologique.

Par conséquent, en prenant aussi en compte les axes d'amélioration, ce programme pourra toujours être repris par une nouvelle équipe dans le cadre d'une étude sur le comportement des populations neuronales et leur implication dans la dynamique du sommeil.

Bibliographie

[Baty, 2018] Baty, H. (2018). Approche numérique à l'usage du physicien pour résoudre les équations différentielles ordinaires.

[Borbely, 1982] Borbely, A. (1982). A two process model of sleep regulation. . *Hum Neurobiol.*

[Brown *et al.*, 2008] Brown, R., McKenna, J., Winston, S., Basheer, R., Yanagawa, Y., Thakkar, M. et McCarley, R. (2008). Characterization of gabaergic neurons in rapideye-movement sleep controlling regions of the brainstem reticular formation in gad67-green fluorescent protein knock-in mice. *Eur J Neurosci 27.*

[Daan *et al.*, 1984] Daan, S., Beersma, D. et Borbely, A. (1984). Timing of human sleep : recovery process gated by a circadian pacemaker. *The American Journal of Physiology.*

[Darnige *et al.*, 2019] Darnige, E., Grimaud, A., Gruel, A. et Kuntz, A. (2019). Developing a computational model of paradoxical sleep.

[Diniz Behn et Booth, 2010] Diniz Behn, C. et Booth, V. (2010). Simulating microinjection experiments in a novel model of the rat sleep-wake regulatory network. *Journal of Neurophysiology.*

[Diniz Behn et Booth, 2011] Diniz Behn, C. et Booth, V. (2011). A population network model of neuronal and neurotransmitter interactions regulating sleep-wake behavior in rodent species. *Springer Series in Computational Neuroscience.*

[Diniz Behn et Booth, 2012] Diniz Behn, C. et Booth, V. (2012). A fast slow analysis of the dynamics of rem sleep. *Society for Industrial and Applied Mathematics.*

[Fleshner *et al.*, 2011] Fleshner, M., Booth, V., B.Forger, D. et Diniz Behn, C. (2011). Circadian regulation of sleep-wake behaviour in nocturnal rats requires multiple signals from suprachiasmatic nucleus. *Philosophical Transactions of The Royal Society.*

[Fuller *et al.*, 2007] Fuller, P., Saper, C. et Lu, J. (2007). The pontine rem switch : past and present. *The Journal of physiology.*

[Iber *et al.*, 2007] Iber, C., Ancoli-Isreal, S., Chesson Jr, A. et Quan, S. (2007). The aasm manual for the scoring of sleep and associated events. *Rules Terminology and Technical Specifications.*

[Joshi *et al.*, 2007] Joshi, A., Salib, M., Viney, T., Dupret, D. et Somogyi, P. (2007). Behavior-dependant activity and synaptic organization of septo-hippocampal gabaergic neurons selectively targeting the hippocampal ca3 area. *Neuron.*

[Lopes da Silva *et al.*, 1974] Lopes da Silva, F., Hoeks, A., Smits, H. et Zetterberg, L. (1974). Model of brain rhythmic activity. the alpha-rhythm of the thalamus. *Kybernetik.*

[Lu *et al.*, 2004] Lu, J., Devor, M. et Saper, C. (2004). A pontine tegmental flip-flop switch for regulation of rem sleep. *Soc Neurosci Abst.*

[Lydic et Baghdoyan, 1993] Lydic, R. et Baghdoyan, H. (1993). Pedunculopontine stimulation alters respiration and increases ach release in the pontine reticular formation. *Journal of Physiology.*

[McCarley et Hobson, 1975] McCarley, R. et Hobson, J. (1975). Neuronal excitability modulation over the sleep cycle. *Science.*

[Moruzzi, 1972] Moruzzi, G. (1972). The sleep-waking cycle. *Ergebnisse der Physiologie.*

[Phillips et Robinson, 2007] Phillips, A. et Robinson, P. (2007). A quantitative model of sleep-wake dynamics based on the physiology of the brainstem ascending arousal system. *J Biol Rhythms.*

[Rasch et Born, 2013] Rasch, B. et Born, J. (2013). About sleep's role in memory. *Physiological Reviews*.

[Rempe *et al.*, 2010] Rempe, M., Best, J. et Terman, D. (2010). A mathematical model of the sleep/wake cycle. *J Math Biol*.

[Saper *et al.*, 2001] Saper, C., Chou, T. et Scammell, T. (2001). The sleep switch : hypothalamic control of sleep and wakefulness. *Trends Neurosci*.

[Schellenberger Costa *et al.*, 2016] Schellenberger Costa, M., Born, J., Claussen, J.-C. et Martinetz, T. (2016). Modeling the effect of sleep regulation on a neural mass model. *Journal of Computational Neuroscience*.

[Tamakawa *et al.*, 2006] Tamakawa, Y., Karashima, A., Koyama, Y., Katayama, N. et Nakao, M. (2006). A quartet neural system model orchestrating sleep and wakefulness mechanisms. *J Neurophysiol*.

[Wilson et & Cowan, 1973] Wilson, H. et & Cowan, J. (1973). A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue. *Kybernetik*.

Appendices

Annexe A

Tableaux des paramètres

Paramètres utilisés pour le modèle rongeur à 3 populations de neurones	**Valeurs modèle à 3 populations à l'état physiologique**
Wmax(Hz)	6.5
Nmax(Hz)	5
Rmax(Hz)	5
τ_W (ms)	25 000
τN (ms)	10 000
τR (ms)	1 000
αN (ms^{-1})	0.25
γG (ms^{-1})	4
αR (ms^{-1})	0.25
γA (ms^{-1})	3
αW (ms^{-1})	0.5
βW (ms^{-1})	-0.23
βR (ms^{-1})	-0.6
γE (ms^{-1})	5
Hmax	1
Θw (ms^{-1})	2
τhw (ms)	600 000
τhs (ms)	380 000
κ	2.5

TABLE A.1 – **Paramètres du modèle rongeurs à 3 populations de neurones.** Les valeurs sont issues de [Diniz Behn et Booth, 2012].

Paramètres utilisés pour les modèles rongeurs à 6 populations de neurones.	**Valeurs du modèle à 6 populations à l'état physiologique.**	**Valeurs du modèle à 6 populations avec lésion du SCN**	**Valeurs du modèle à 6 populations, adapté aux injections**
$g_{N,LC}$ (ms^{-1})	1.5	1.5	1.5
$g_{A,LC}$ (ms^{-1})	3.5	3.5	3.5
$g_{S,DR}$ (ms^{-1})	1.5	1.5	1.5
$g_{A,DR}$ (ms^{-1})	3.5	3.5	3.5
$g_{A,R}$ (ms^{-1})	2.5	2.5	2.5
$g_{N,R}$ (ms^{-1})	3.5	3.5	3.5
$g_{S,R}$ (ms^{-1})	3.5	3.5	3.5
$g_{A,WR}$ (ms^{-1})	1	1	1
$g_{G,WR}$ (ms^{-1})	1.7	1.7	1.7
$g_{G,VLPO}$ (ms^{-1})	0.5	0.5	0.5
$g_{A,SCN}$ (ms^{-1})	0.2	0.2	0.2
$g_{S,SCN}$ (ms^{-1})	0.2	0.2	0.2
$g_{G,LC}$ (ms^{-1})	1.5	1.5	1.5
$g_{G,DR}$ (ms^{-1})	1.5	1.5	1.5
$g_{G,R}$ (ms^{-1})	1.25	1.25	1.25
$g_{N,VLPO}$ (ms^{-1})	2	2	2
$g_{S,VLPO}$ (ms^{-1})	2	2	2
$g_{G(SCN),LC}$ (ms^{-1})	4	0	4
$g_{G(SCN),DR}$ (ms^{-1})	4	0	4
$g_{G(SCN),VLPO}$(ms^{-1})	1.8	0	1.8
$g_{G(SCN),R}$ (ms^{-1})	0.3	0	0.3
τ_{DR} (ms)	25 000	25 000	25 000
τ_{R} (ms)	1 000	1 000	1 000
τ_{WR} (ms)	10 000	10 000	10 000
τ_{VLPO} (ms)	10 000	10 000	10 000
τ_{SCN} (ms)	500	500	500
τ_{N} (ms)	25 000	25 000	25 000
τ_{S} (ms)	25 000	25 000	25 000
$\tau_{A(R)}$ (ms)	10 000	10 000	10 000
$\tau_{A(WR)}$ (ms)	10 000	10 000	10 000
τ_{G} (ms)	10 000	10 000	10 000
$\tau_{G(SCN)}$ (ms)	10 000	10 000	10 000
LCmax(Hz)	6.5	6.5	6.5
DRmax(Hz)	6.5	6.5	6.5
Rmax(Hz)	5	5	5
WRmax(Hz)	5	5	5
VLPOmax(Hz)	5	5	5
SCNmax(Hz)	8	8	8

TABLE A.2 – **Paramètres des modèles rongeurs à 6 populations de neurones.** Les valeurs sont issues de [Fleshner *et al.*, 2011] pour le modèle à l'état physiologique ainsi que pour le modèle comportant une lésion. Les valeurs du modèle à injections sont tirées de [Diniz Behn et Booth, 2010]. Les valeurs de couleur rouge permettent de mettre en évidences les différences avec le modèle à l'état physiologique.

Paramètres utilisés pour les modèles rongeurs à 6 populations de neurones.	**Valeurs du modèle à 6 populations à l'état physiologique.**	**Valeurs du modèle à 6 populations avec lésion du SCN**	**Valeurs du modèle à 6 populations, adapté aux injections**
γ_A (ms^{-1})	3	3	3
γ_N (ms^{-1})	5	5	5
γ_S (ms^{-1})	5	5	5
γ_G (ms^{-1})	4	4	4
$\gamma_{G(SCN)}$ (ms^{-1})	4	4	4
α_{LC} (ms^{-1})	0.75	0.75	0.75
α_{DR} (ms^{-1})	0.75	0.75	0.75
α_R (ms^{-1})	0.25	0.25	0.25
α_{WR} (ms^{-1})	0.25	0.25	0.25
α_{VLPO} (ms^{-1})	0.25	0.25	0.25
α_{SCN} (ms^{-1})	1.5	1.5	1.5
β_{LC} (ms^{-1})	-1.85	-1.85	-1.85
β_{DR} (ms^{-1})	2	2	2
β_{WR} (ms^{-1})	-0.2	-0.2	-0.2
β_R (ms^{-1})	-0.82	-0.82	-0.82
β_{SCN} (ms^{-1})	0.25	0.25	0.25
ΘLC (ms^{-1})	3	3	3
$\tau_{h(s)}$ (ms)	200 000	200 000	200 000
Hmax	1.1	1.1	1.1
$\tau_{h(w)}$ (ms)	700 000	700 000	700 000
κ	2.5	2.5	2.5
G_{min}	-	-	0.3
G_{max}	-	-	2.25
$P_{G(0)}$	-	-	2
$Q_{G(0)}$	-	-	1
A_{min}	-	-	0.3
A_{max}	-	-	2
$P_{A(0)}$	-	-	0.8
$Q_{A(0)}$	-	-	0.55

TABLE A.3 – **(Suite) Paramètres des modèles rongeurs à 6 populations de neurones.** Les valeurs sont issues de [Fleshner *et al.*, 2011] pour le modèle à l'état physiologique ainsi que pour le modèle comportant une lésion. Les valeurs du modèle à injections sont tirées de [Diniz Behn et Booth, 2010]. Les valeurs de couleur rouge permettent de mettre en évidences les différences avec le modèle à l'état physiologique.