Amélioration de l'Interface utilisateurs du modèle FarmSim

Marine Blanchard, Romane Noyer

Institut Supérieur d'Informatique de Modélisation et de leurs Applications

Filière Systèmes d'Information et d'Aide à la Décision

Projet de 2eme année

Amélioration de l'Interface utilisateurs du modèle FarmSim

Auteurs :
Mme Marine Blanchard
Mme Romane Noyer

Encadrants :
M. Raphaël Martin
Dr. Philippe Lacomme

11 mars 2018

Remerciements

Nous souhaitons en premier lieu remercier chaleureusement les différentes personnes présentes sur ce projet qui ont pu nous aider à développer celui-ci.

Nous commençons avec M. Raphaël MARTIN, notre tuteur de projet et travaillant à l' INRA, pour son accueil bienveillant, sa très grande disponibilité et son suivi au fil des mois. Son aide précieuse et sa force de proposition ont su nous aiguiller et nous débloquer à de nombreux moments.

Nous tenons également à remercier M. Philippe LACOMME, notre tuteur ISIMA, qui a veillé au bon déroulement de notre projet. Nous remercions l'UREP, qui nous a prêté ses locaux et son équipe qui a accepté de se libérer pour nous écouter et nous conseiller pour la présentation orale de notre projet.

Enfin, nos remerciements vont vers la structure de l'ISIMA, qui a su nous encadrer, nous donner les outils ou les locaux pour que notre projet se déroule dans les meilleures conditions.

Table des figures

Résumé

Actuellement, nous sommes au cœur d'une période charnière dans laquelle l'impact environnemental des pratiques agricoles est de plus en plus pris en compte dans le secteur de l'agriculture, suite à une prise de conscience plus générale sur l'état de notre planète et sur la qualité des récoltes produites. De nombreux facteurs (fertilisations, labour, type de semence, ...) influent sur la qualité des sols ainsi que sur les émissions de gaz à effet de serre. Afin de mieux évaluer ces impacts et ainsi mieux adapter nos exploitations agricoles, l'Institut National de Recherche Agronomique (**INRA**) développe de nombreux outils dont **FarmSim** qui a été mis au point à l'**UREP**.

FarmSim est un logiciel effectuant des simulations sur des exploitations de type polyculture-élevage. Les résultats permettent de calculer de nombreuses sorties les (émission de gaz à effet de serre, rendement, stockage de carbone, ...) en fonction de nombreux paramètres (entrées) et sur de nombreuses années. Notre projet a eu pour objectif d'améliorer la présentation des résultats. En effet, l'**affichage** existant n'étant plus adapté suite aux différentes modifications du logiciel, nous avons dû **optimiser** les **graphiques** proposés à l'utilisateur pour améliorer son expérience. Nous avons pour cela utilisé la librairie **JFreeChart**, une bibliothèque **JAVA** permettant de créer différents types de graphiques. Nous avons également amélioré la création de ces graphiques en permettant à l'**utilisateur** de choisir quelles variables il souhaite visualiser sur chaque exploitation.

Mots clés : INRA, UREP, JAVA, JFreeChart, FarmSim, Affichage, Optimisation, Graphique, Utilisateur.

Abstract

Nowadays, we are at the heart of a pivotal period in which the environmental impact of agricultural practices is increasingly taken into account in the agriculture sector, following a great general awareness of the state of our planet and the quality of the crops produced. Many factors (fertilization, plowing, type of seed, etc.) affect soil quality as well as greenhouse gas emissions. To better assess these impacts and adapt our farms better, National Institute of Agricultural Research (**INRA** in French) develops many tools including **FarmSim**, which was developed at **UREP** service.

FarmSim is a software that performs simulations on mixed crop-livestock farms. The results make it possible to calculate many outputs (greenhouse gas emissions, yield, carbon storage, etc.) according to numerous parameters (inputs) and for many years. Our project aimed to improve the presentation of the results. Indeed, the existing **display** is no longer suitable following the various changes in the software, we had to **optimize** the **graphics** offered to the user to improve its experience. We have used the **JFreeChart** library, a **JAVA** library to create different types of graphics. We also improved the creation of these graphs by allowing the **user** to choose which variables he wants to visualize on each farm.

Key Words : INRA, UREP, JAVA, JfreeChart, FarmSim,Display, Optimize, Graphics, User.

Table des matières

Introduction

L'agriculture française contribue pour près d'un cinquième aux émissions de gaz à effet de serre (GES). Il est donc essentiel pour les chercheurs de mieux comprendre les processus générant ces GES afin de déterminer et analyser les différentes options portant sur des pratiques agricoles et susceptibles de réduire ses émissions de GES, comme la réduction des émissions de protoxyde d'azote (N2O, puissant gaz à effet de serre produit par la transformation des engrais ou des déjections animales dans les sols cultivés) et de méthane (CH4, gaz à effet de serre provenant notamment des élevages). Dans un contexte comme celui-ci, le logiciel FarmSim est un puissant outil d'analyse pour ces recherches.

FarmSim est un logiciel qui simule l'activité d'une ferme sur une durée donnée, fournissant la quantité de GES produit par celle-ci en fonction des différents paramètres donnés en entrées.

Actuellement, les données sons fournies sous la forme d'un fichier CSV, et l'interface des résultats existante est devenue obsolète suite aux diverses modifications successives du logiciel. Le but de ce projet est de rendre à nouveaux fonctionnel la génération des graphique puis d'améliorer cet affichage afin qu'il soit d'une part plus simple à utiliser, mais aussi qu'il soit plus complet et mieux adapté au besoin de l'utilisateur.

Chapitre 1

Présentation

1.1 Présentation de la Structure

L'INRA (Institut National de Recherche Agronomique) est un organisme public français de recherche en agronomie fondé en 1946 sous la tutelle du ministère chargé de la Recherche et de l'Agriculture.
Il s'agit du premier institut de recherche agronomique en Europe et deuxième dans le monde en nombre de publications en sciences agricoles, de la plante et de l'animal. L'INRA mène des recherches pour améliorer la qualité de notre alimentation et pour une agriculture durable et respectueuse de l'environnement.

Elle se répartit en 13 départements de recherche et 17 centres dans la France entière, avec 250 Unités de Recherche (UR) ou Unité Mixte de Recherche. Nous avons tout particulièrement travaillé avec l'une de ces dernières : l'UREP.
Le projet de l'Unité Mixte de Recherche sur l'Ecosystème Prairial (UREP) porte sur " l'écologie, le fonctionnement et les services de la prairie permanente dans un contexte de changement global ".
L'UREP possède une grande expertise dans le domaine de l'écologie et plus particulièrement sur l'impact du changement climatique, des gaz à effet de serre, de la séquestration du carbone dans le sol, ou encore des pratiques de gestion sur la dynamique d'une prairie.

1.2 Présentation du contexte

Malgré de nombreux efforts dans le secteur (une baisse de 6 pourcents entre 1990 et 2013), l'Agriculture génère 16 pourcents des émissions de gaz à effets de serre comme le N2O, puissant gaz à effet de serre produit par la transformation des engrais ou des déjections animales dans les sols cultivés) ou bien le méthane CH4, gaz à effet de serre provenant notamment des élevages.

Avec comme objectif de continuer à atténuer ces émissions de GES responsable des chan-

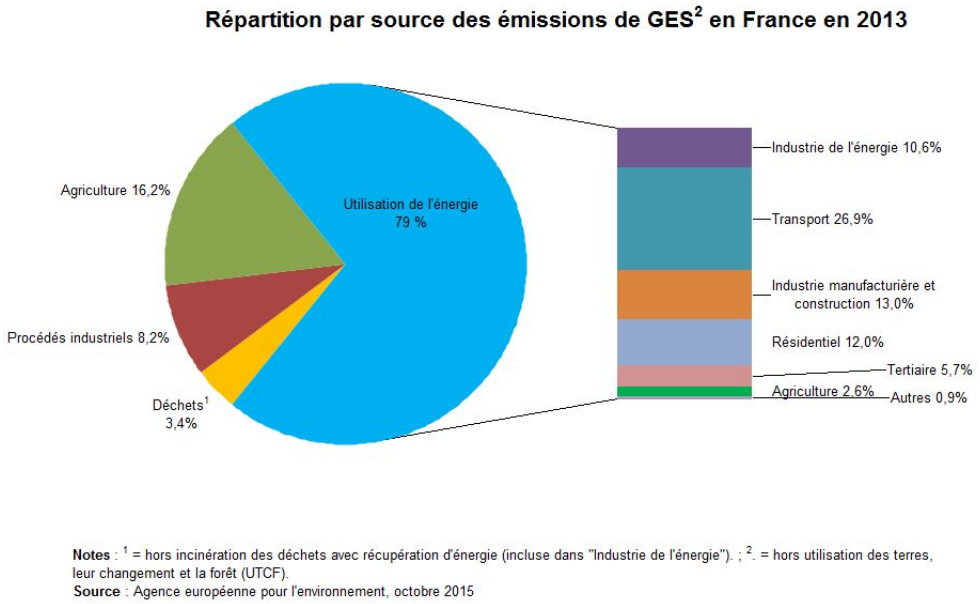


FIGURE 1.1 – Répartition des émissions de GES en 2016 en France

gements climatiques récents, la France et l'Europe encourage financièrement différentes recherches afin de mieux comprendre les mécanismes responsables de cette émission de GES, et de mettre en place de nouvelles solutions qui permettront à l'avenir de faire baisser à nouveau ce chiffre lié à l'Agriculture.

1.3 Présentation de FarmSim

FarmSim est un logiciel de simulation qui pour une exploitation agricole donnée, calcule la quantité des différents gaz à effet de serre émis par celle-ci. Il s'appuie principalement sur deux autres logiciels que nous détailleront après, qui sont PaSim et CERES-EGC, ainsi qu'un troisième modèle que nous ne détaillerons pas ici.

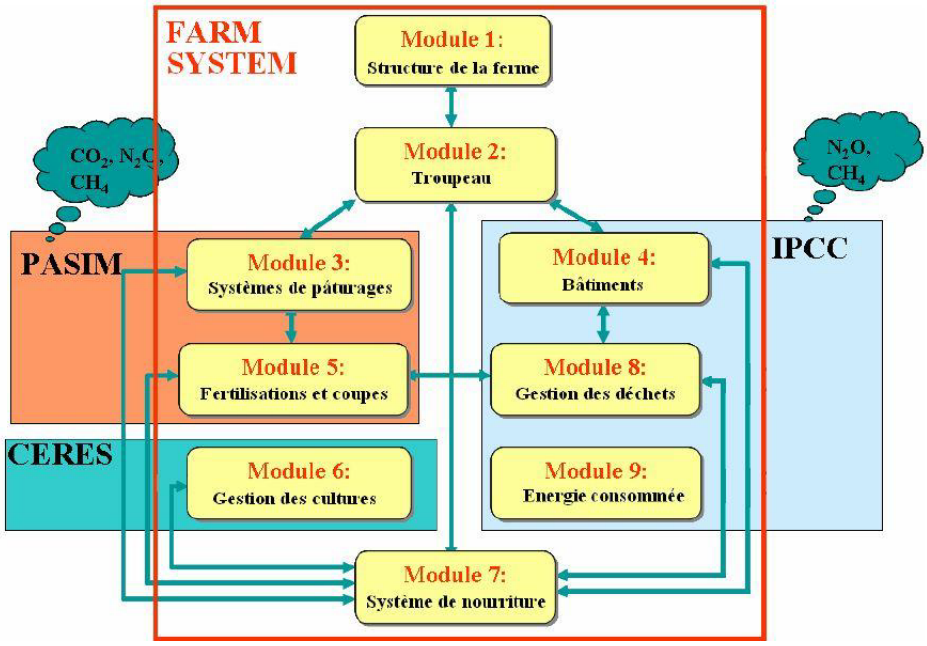


FIGURE 1.2 – Fonctionnement du logiciel FarmSim

FarmSim a été développé à l'origine en Visual Basic en 2002 dans le cadre du projet européen GreenGrass. En 2007, son code fut changé pour passer au langage de programmation objet JAVA, avant de subir de nombreux modifications au fil des années jusqu'à aujourd'hui. FarmSim est un outil permettant de simuler une ferme de polyculture-élevage et d'en modéliser tous les flux de GES.

Plusieurs objectifs découlent de ce logiciel :
Mettre en évidence les différents gaz à effet de serre émis par ces fermes.
Structurer pour une ferme donnée l'ensemble des données potentiellement utiles au calcul du bilan des GES sur une échelle de temps donnée.
Décrire de manière précise tous les différents types d'exploitation agricole.

1.4 Présentation de PASIM

PaSim (Pasture Simulation model) est un modèle de simulation d'un écosystème prairial à l'échelle de la parcelle. Il simule les différents flux de carbone, d'azote, d'eau et d'énergie au niveau de l'interface « sol-plante-animal-atmosphère ». Ce modèle prend un grand nombre d'entrées que l'on peut séparer en 5 catégories : climat, sol, végétaux, animaux et gestion. Il donne en sortie des données sur les flux de GES, les productions animales, ... Il a été programmé en FORTRAN 90 et est intégré au cœur de FarmSim sous forme d'un exécutable. FarmSim a donc pour rôle de formater les entrées puis de fournir à l'exécutable ces dernières dans un format que PaSim peut intérpréter. Il récupère ensuite, une fois les simulations exécutées, les fichiers de sortie contenant les résultats pour ensuite les mettre en forme dans un format plus facilement compréhensible par l'utilisateur.

1.5 Présentation de CERES-EGC

CERES-EGC est issu du modèle d'aide à la décision de culture CERES, développé dans les années 1980 aux USA. Il l'adapte afin de simuler les impacts environnementaux des cultures. En effet, il modélise la croissance et le développement de la plante, les transferts de chaleurs, d'eau, de carbone, de nitrate etc. A l'instar de PaSim, il a été écrit en FORTRAN 90, et récupère ses données de la même manière : FarmSim envoie et compresse ses fichiers en entrée de CERES-EGC puis récupère et interprète les fichiers de sortie que celui-ci lui renvoie. CERES-EGC modélise les différents flux de CO2, N2O, NH3* entre la terre et l'atmosphère via divers processus comme la biotransformation.

Chapitre 2

Réalisations et Conception

2.0.1 Diagrammes de Gantt

Avant même de commencer le projet à proprement parler, il nous a fallu mettre en place une organisation sous forme de diagramme, afin de clarifier nos objectifs, découper les différentes étapes et estimer dans le temps la durée de ceux-ci.
(Voir Annexes 1 pour le Diagramme prévisionnel et 2 pour le Diagramme réel)

Plusieurs grandes étapes évidentes en sont ressorties, comme par exemple l'approfondissement du sujet et l'apprentissage du langage JAVA en premier lieu, la réalisation de la solution et l'intégration de celle-ci au logiciel existant.
Bien sûr, il est évident que la réalisation effective de cet emploi du temps a été quelque peu différente, et ce pour de nombreuses raisons : sous-estimation du temps d'apprentissage, nombreux blocages, organisation personnelle avec les autres matières... De nombreux facteurs font que nous avons parfois pris du retard, ou au contraire été plus rapides que prévu.

En effet, plusieurs points sont importants à aborder :
Tout d'abord l'apprentissage du JAVA. Nous avions un peu naïvement prévu une courte période en début de projet afin d'apprendre les notions liées à notre projet, puis de commencer celui-ci. Or il se trouve que bien naturellement, surtout en tant qu'étudiantes, nous ne nous arrêtons jamais d'apprendre. Lors du projet, nous avons appris " sur le tas", chaque nouveau problème nous faisant apprendre de nouvelles compétences, et d'autant plus vers la fin où nous avons eu à explorer de nouveaux moyens. Il est donc logique que cette tâche s'étale sur l'intégralité de notre projet.
La création de graphe est elle aussi étalée sur une période plus longue, et ce pour deux principales raisons : notre organisation personnelle et le travail de fin de projet. Suite à des partiels et à d'autres projets que nous avons priorisé face à celui-ci, nous avons retardé le début de notre travail sur la création de graphe, ce qui a donc retardé également toutes

les autres étapes qui suivaient. Une fois notre travail terminé et alors que nous étions sur une autre phase, une nouvellle fonctionnalité souhaitée par notre tuteur a quelque peu chamboulé l'architecture que nous avions mise en place pour la création de notre graphe. Nous sommes donc revenues sur cette partie et avons donc modifié en conséquence notre code pour nous adapter à cette nouvelle exigence. Nous avons aussi rajouté des étapes qui n'étaient pas présentes car elles ont émergé en cours de projet, n'étant donc pas d'actualité au début de celui-ci.

Nous pouvons constater un emballement du projet sur la fin, déjà premièrement car l'échéance approchait, mais également car notre emploi du temps s'éclaircissait, nous accordant donc plus de temps à consacrer à l'avancement du projet. Au final, nos objectifs initiaux sont remplis, et malgré un retard en début de projet, nous avons su compenser en étant plus efficaces sur d'autres parties et en nous concentrant sur notre travail.

2.1 Conception

Cette section est destinée à la présentation du travail effectué, la justification de nos différentes technologies, ainsi que la restitution de notre résultat final, le tout expliqué et commenté de notre expérience.
Introduction du travail à effectuer :
Dans un premier temps, nous allons présenter les grandes étapes de notre projet, qui ont été la création d'un graphique avec la bibliothèque JFreeChart de manière indépendante, puis son amélioration ; l'intégration de noter solution au logiciel en l'adaptant à son architecture et au travail précédemment accompli, et l'amélioration de l'interface des résultats afin qu'elle soit plus facilement exploitable par les futurs utilisateurs.

2.1.1 Création et amélioration d'un graphe

Aux vues des différents bugs dans le logiciel FarmSim dû à de nombreux paramètres extérieurs, nous avons préféré créer notre solution de manière indépendante.
Il nous fallait créer un graphe à partir d'un fichier CSV regroupant toutes les données d'une même ferme, afin de pouvoir interpréter ces données. Pour cela, nous devions lire ce fichier pour le mettre dans une matrice, puis lire cette matrice (chaque colonne représentant les différentes variables) pour proposer à l'utilisateur de choisir le paramètre qui représentera l'axe Y de notre graphique. Puis, créer le graphique et l'afficher selon nos consignes (l'axe X temporel devant représenter à la fois les années mais également les jours sur une année). Enfin, il nous fallait améliorer celui-ci pour qu'il soit plus visible et surtout plus adapté aux situations réelles : Ajout de jalons d'informations, superposition de plusieurs tracés sur un même graphe... C'est la partie centrale de notre projet.

2.1.2 Intégration de la solution au logiciel FarmSim

Une autre partie primordiale à la réussite de notre projet consistait en l'incorporation de notre code à celui déjà existant, en substituant et fusionnant tout le code lié à la création du graphique existant, puis en incluant nos propres fonctions de lecture et création de graphe dans le reste de FarmSim (à modifier avec le screenshot).

2.1.3 Amélioration de l'interface Résultats

Certaines fonctionnalités étant devenues obsolètes / inutiles, il nous fallait adapter notre interface en fonction des nouveaux besoins, et enlever ce qui était définitivement inutile (par exemple le choix entre un graphique de type Crop ou de Type Grass, qui n'a plus lieu d'être à présent avec notre nouvelle interface de création de graphique). (à compléter avec les screenshots).

(Améliorer l'exportation PDF en corrigeant le code qui prenait en paramètre une liste de graphes)

2.1.4 Présentation des technologies

Eclipse

Eclipse est un environnement de développement (IDE) principalement dédié au langage JAVA mais également disponible pour d'autres langages de programmation.

Il intègre de nombreuses bibliothèques ainsi que la machine virtuelle Java ; énormément d'outils permettant d'aider le programmeur à coder, compiler, ou encore debugger.
Son architecture est développée autour de la notion de plugin : cela signifie que toutes ses fonctionnalités sont développées en tant que plugins, si nous souhaitons rajouter une fonctionnalité, il nous suffit de télécharger le plugin associé et de l'intégrer à Eclipse. Il rend donc notre environnement de développement modulable et personnalisable à souhait, sans pour autant encombrer de fonctionnalités inutiles.
Il centralise donc de nombreux outils et simplifie leur mise en oeuvre ce qui apporte un énorme gain de temps pour le développeur.

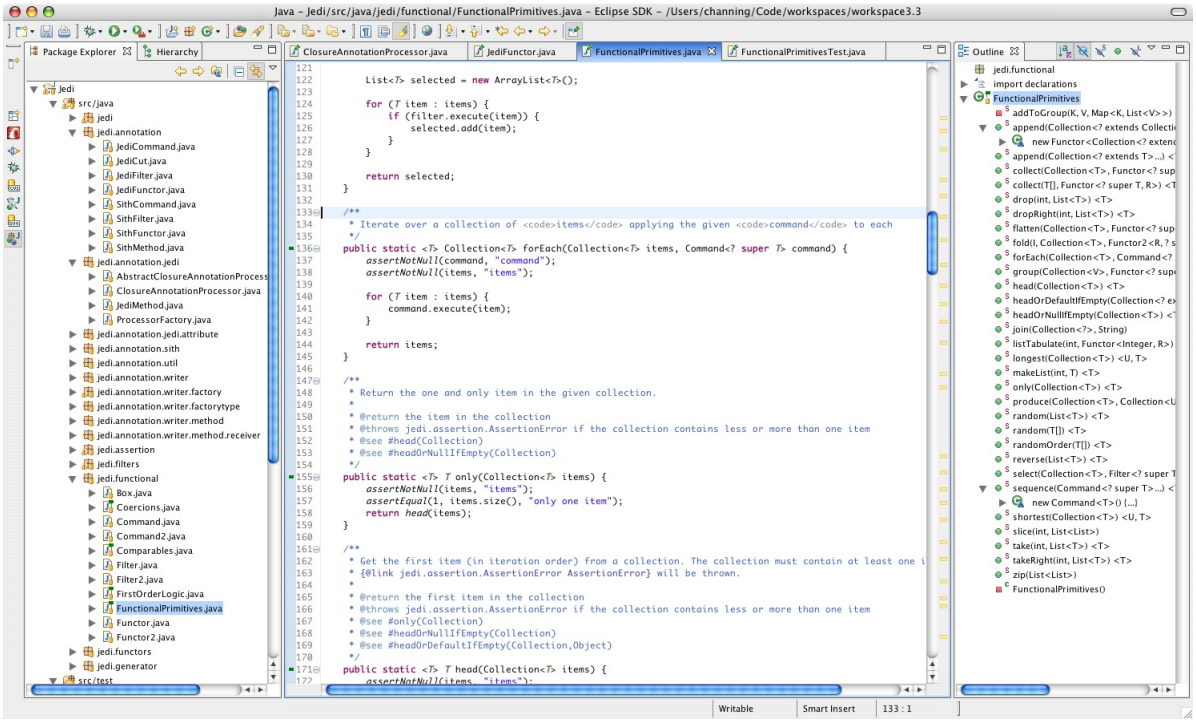


FIGURE 2.1 – Capture d'écran de l'environnement Eclipse

Swing

Swing est une bibliothèque graphique créée pour JAVA. Il constitue une ds principales améliorations apportées par JAVA 2 par rapport aux anciennes versions. Par rapport à AWT (L'ancienne bibilothèque graphique développée), les interfaces graphiques créées sont portables, c'est à dire identiques peu importe le système d'exploitation, le revers de la médaille étant que les performances en sont réduites.

Son architecture utilise le pattern Modèle-Vue-Contrôleur(MVC) : l'utilisateur fait une action en appelant le contrôleur. Celui-ci appelle à son tour le modèle, et ce dernier notifie la vue qu'elle doit se rafraîchir en interrogeant le modèle. (screenshot) Cette architecture permet une séparation des données et du contrôleur, une évolution mieux maîtrisée ainsi qu'une plus grande souplesse dans le développement.

JFreeChart

JFreeChart est une API Java open source et gratuite qui permet d'afficher des données statistiques sous la forme de graphiques. Elle possède plusieurs formats dont le camembert, les barres ou les lignes et propose de nombreuses options de configuration pour personnaliser le rendu des graphiques. Elle peut s'utiliser dans des différents typpes d'applications et permet également d'exporter le graphique sous la forme d'une image sous de nombreux formats (PDF, PNG, JPEG, SVG...).
Elle comprend une documentation fournie qui la rend assez facile d'utilisation, et peut être utilisée à la fois côté client et côté serveur.

Plusieurs notions sont intéressantes à comprendre lorsque l'on crée un graphique JFreeChart : Series –> C'est la courbe. On lui associera des données afin de pouvoir la tracer. Dataset –> C'est le graphique en tant que tel, on lui donne des axes et un ou plusieurs series. Chart –> C'est le graphique "étendu". Il comprend un ou plusieurs DataSet, un titre, des unités et des légendes.

2.2 Conception de la solution

2.2.1 Création du graphe

L'Onglet "Résultats" étant actuellement non fonctionnel suite à de nombreux bugs, nous avons souhaité développer nos fonctions de façon isolée sur Eclipse.
Pour cela, nous avons téléchargé un des fichiers CSV d'une ferme (le fichier en paramètres rassemblant toutes les données à afficher afin de le lire en dur).

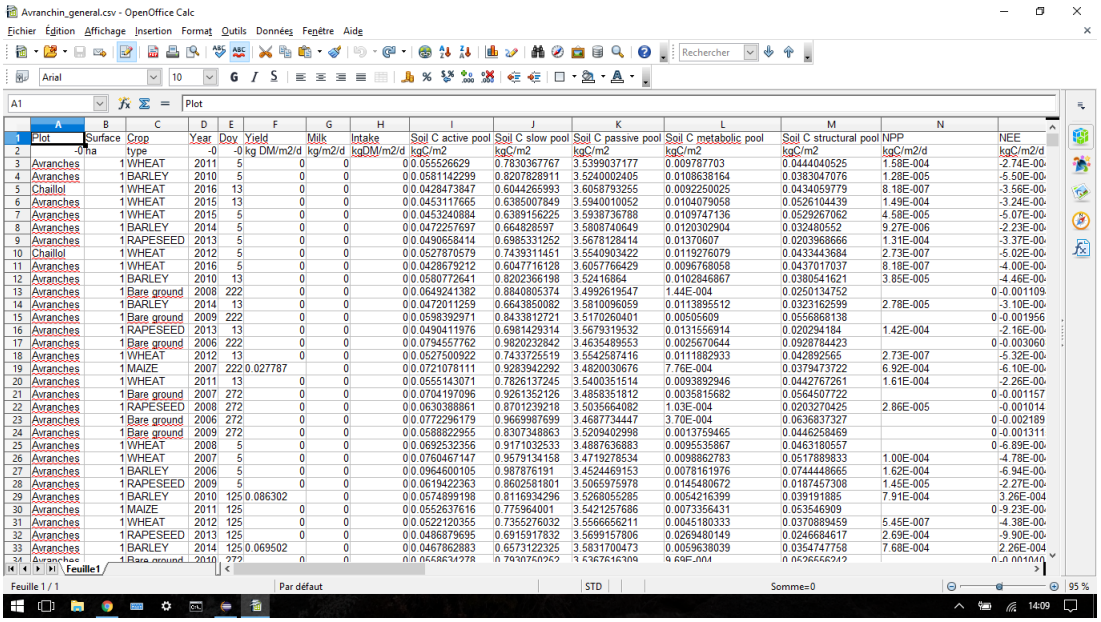


FIGURE 2.2 – Capture d'écran du fichier CSV de la ferme Avranchin

Pour faire un graphique fonctionnel, nous avons besoin :
- d'une matrice de données triée–> C'est le tableau qui va contenir toutes les données à traiter, que nous devrons trier par année puis par jour.
-d'une matrice de variables –> cette matrice contiendra toutes les variables du fichier csv ainsi que l'unité correspondante, afin de permettre à l'utilisateur de choisir quelle variable il souhaite afficher dans son graphique. Une correspondance entre la variable et ses données associées sera faite.

La figure 2.3 synthétise les différentes listes et tableaux créées et à quel endroit vont être stockées les données avant d'être exploitées.

Pour construire ces matrices, différentes étapes sont nécessaires. En premier lieu, nous devons créer une fonction de lecture : celle-ci sert à parcourir notre ferme en entrée et remplir au fur et à mesure une liste de tableau de chaînes de caractères contenant chaque

header

Plot	Surface	Crop	Year	Doy	Yield	Milk
-0	ha	type	-0	-0	kg DM/m2/d	kg/m2/d
Avranches	1	WHEAT	2011	5	0	0
Avranches	1	BARLEY	2010	5	0	0
Chaillol	1	WHEAT	2016	13	0	0
Avranches	1	WHEAT	2015	13	0	0
Avranches	1	WHEAT	2015	5	0	0
Avranches	1	BARLEY	2014	5	0	0
Avranches	1	RAPESEED	2013	5	0	0
Chaillol	1	WHEAT	2012	5	0	0

fileData

matrix

FIGURE 2.3 – Schéma du fichier CSV et du stockage de leur données

ligne de la ferme.

Nous mettons les 2 premieres lignes de cette liste de tableau de String dans une matrice nommée "header" qui sera notre matrice de valeur.
le reste se stocke dans une liste de chaînes de caractères fileData dont chaque tableau correspond à une ligne et chaque élément du tableau correspond à une cellule/valeur du fichier lu.

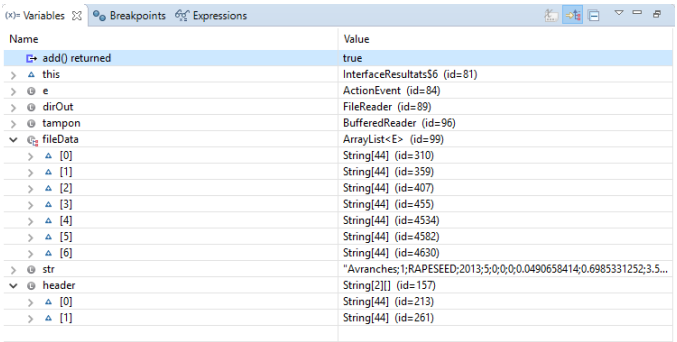


FIGURE 2.4 – Capture d'écran de la matrice "header" et de la liste "fileData"

La figure 2.4 montre la structure de notre matrice et de notre liste, contenant chacune des chaînes de caractères.
Sur la figure 2.5, on peut aperçevoir la liste fileData : les chaînes de caratères contenues dans fileData[0][] correspondent à la première ligne de données du fichier CSV. Afin de

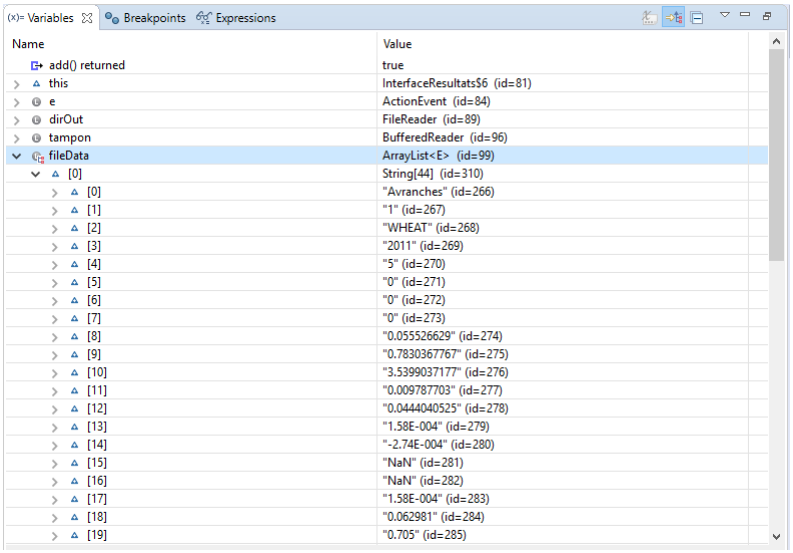


FIGURE 2.5 – Capture d'écran de la liste "fileData"

rendre ces données exploitables par le graphique, nous trions notre liste par année puis par jour, rendant notre tableau trié par ordre chronologique (notre but final étant d'interpréter des données selon leur évolution dans le temps).

Lors d'une phase d'amélioration ultérieure, nous avons rajouté un second tri qui conserve notre tri chronologique et qui sépare et tri par ordre alphabétique selon les différentes parcelles d'une même ferme.

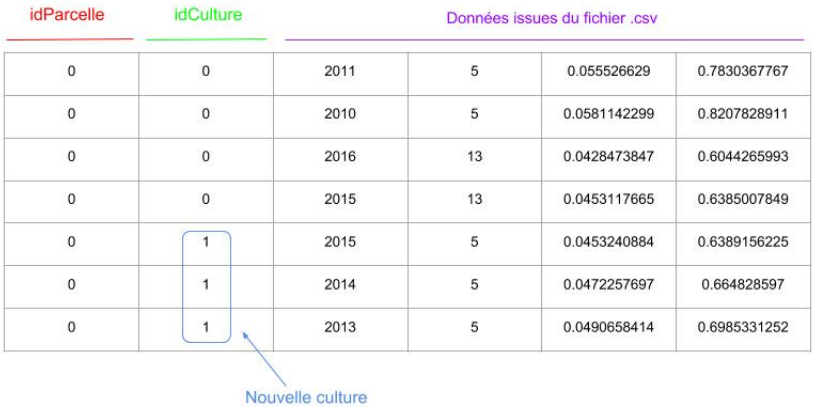
```
// Trie de la liste suivant les annees puis les jours
Collections.sort(fileData, new Comparator<String[]>() {
    @Override
    public int compare(String[] o1, String[] o2) {
    int day1 = Integer.valueOf(o1[3])*365+Integer.valueOf(o1[4]);
    int day2 = Integer.valueOf(o2[3])*365+Integer.valueOf(o2[4]);
    return day1-day2;
    }
});

// Trie de la liste suivant les parcelles
Collections.sort(fileData, new Comparator<String[]>() {
    @Override
    public int compare(String[] o1, String[] o2) {
    return o1[0].compareTo(o2[0]);
    }
});
```

FIGURE 2.6 – Capture d'écran du tri par années et du tri par parcelle

Notre "dataFile" trié, nous devons à présent construire notre matrice de données : Nous effectuons un parcours de notre liste "dataFile" et nous ajoutons ligne par ligne nos données, en prenant bien soin d'enlever les 3 premières colonnes qui contiennent des informations qui ne nous intéressent pas.

Dans un second temps, lors d'une phase d'amélioration que nous développerons ultérieurement, nous avons modifié la structure de notre matrice pour lui rajouter deux colonnes : une colonne d'identifiants de parcelle et une d'identifiants de culture, qui serviront pour l'affichage du graphique.



0	0	2011	5	0.055526629	0.7830367767
0	0	2010	5	0.0581142299	0.8207828911
0	0	2016	13	0.0428473847	0.6044265993
0	0	2015	13	0.0453117665	0.6385007849
0	1	2015	5	0.0453240884	0.6389156225
0	1	2014	5	0.0472257697	0.664828597
0	1	2013	5	0.0490658414	0.6985331252

FIGURE 2.7 – Schéma de la matrice modifiée, avec ajout des idParcelle et idCulture

Pour lister nos parcelles et nos cultures, nous avons créé deux fonctions aux structures similaires :

On crée deux variables : prec et cour, qui nous servirons de comparaison lors de notre parcours ;
On crée ensuite une liste de tableau de chaînes de caractères "LinkId" : Elle effectuera la correspondance entre un identifiant et le nom qui lui est associé (exemple : l'identifiant 2 correspondra à la parcelle " Chaillol") ;
On parcoure notre matrice : lorsque nos deux variables cour et prec sont identiques(c'est à dire que la parcelle/culture que l'on parcoure est identique à celle d'avant), on ajoute notre identifiant actuel à la matrice tandis que lorsqu'elles sont différentes (c'est à dire que l'on change de type de culture/parcelle), on incrémente notre identifiant, puis on l'ajoute dans notre liste "linkId" avant de l'ajouter à notre matrice ;
Enfin, nous mettons à jour notre prec pour qu'il corresponde à notre identifiant cour ;

A partir de notre matrice de variables "header", nous créons une liste dans une combo-

```
// Creation matrice
matrix = new double[nbligne][nbcolonne+2];

// Liste des parcelles
String parcellePrec = null;
int idCour = -1;
parcelleLinkId = new ArrayList<String>();
for(int i=0; i<nbligne; ++i) {
    String parcelleCour = fileData.get(i)[0];
    if(parcelleCour.equals(parcellePrec)) {
        matrix[i][0] = idCour; // Ajout des identifiants des parcelles
    }
    else {
        matrix[i][0] = ++idCour;
        parcelleLinkId.add(parcelleCour);
    }
    parcellePrec = parcelleCour;
}

// Liste des cultures
String culturePrec = null;
idCour = -1;
cultureLinkId = new ArrayList<String>();
for(int i=0; i<nbligne; ++i) {
    String cultureCour = fileData.get(i)[2];
    if(cultureCour.equals(culturePrec)) {
        matrix[i][1] = idCour; // Ajout des identifiants des cultures
    }
    else {
        matrix[i][1] = ++idCour;
        cultureLinkId.add(cultureCour);
    }
    culturePrec = cultureCour;
}

// Ajout des donnees numerique
for(int i=0; i<nbligne; ++i) {
    String [] ligne = fileData.get(i);
    for(int j=3; j<nbcolonne; ++j) {
        matrix [i][j-3+2]= Double.parseDouble(ligne[j]);
    }
}
```

FIGURE 2.8 – Capture d'écran des fonctions de création d'une matrice

box donnant la possibilité à l'utilisateur de choisir quelle variable il souhaite prendre pour la création de son graphique.
Le choix de cette variable va faire correspondre une colonne de notre matrice de données, que l'on mettra en paramètres pour la création de notre graphique.

Maintenant que nos paramètres sont créés et triés, nous pouvons enfin créer notre graphique avec la bibliothèque JFreeChart :

On crée un graphe de type XYChart (Graphique qui affichera une courbe selon les 2 axes)

Pour cela, on fait coïncider notre axe Y (l'axe des ordonnées) avec notre variable mise en paramètres : celle choisie par l'utilisateur. Pour l'axe X (celui des abscisses), c'est un peu plus compliqué. Afin de faire comprendre à JFreeChart que nos valeurs correspondent à des dates, il faut les mettre sous forme de calendrier. C'est seulement une fois cette action effectuée que l'on peut associer nos valeur jour, mois et année de type calendrier à notre axe des abscisses. On peut ensuite créer notre DataSet, puis notre Chart.

InterfaceResultats.java *Graphe.java

```
119          int doy = (int) matrix[i][3];
120          Calendar cal = Calendar.getInstance();
121          cal.set(Calendar.DAY_OF_YEAR, doy);
122          int day = cal.get(Calendar.DATE);
123          int month = cal.get(Calendar.MONTH)+1;
124          int year = (int) matrix[i][2];
125          timeSeriesCour.add(new Day(day, month, year),matrix[i][variable]);
126
127          parcellePrec = parcelleCour;
128          culturePrec = cultureCour;
129         }
130
131         // Creation du dataset
132         TimeSeriesCollection dataset = new TimeSeriesCollection();
133         for(TimeSeries ts : series) {
134            dataset.addSeries(ts); // Ajout des series au dataset
135         }
136
137         xylineChart = ChartFactory.createTimeSeriesChart(chartTitle ,"Time" , legendeY, dataset, true, true, false);
138         ChartPanel chartPanel = new ChartPanel( xylineChart );
139         final XYPlot plot = (XYPlot) xylineChart.getPlot( );
140         chartPanel.setBackground(Color.BLUE);
141
142         XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer( );
143         renderer.setBaseStroke(new BasicStroke(2F));
144         renderer.setAutoPopulateSeriesStroke(false);
145         plot.setRenderer(renderer);
146         plot.setDomainCrosshairVisible(true);
147         plot.setRangeCrosshairVisible(true);
148         setContentPane(chartPanel);
```

FIGURE 2.9 – Capture d'écran des fonctions de création d'un graphe

2.2.2 Intégration des fonctions au logiciel

Une étape importante du projet était d'intégrer notre code à celui déjà existant dans FarmSim.
La solution actuelle (qui n'est plus opérationnelle) utilise une structure qui n'est plus du tout adaptée aux besoins actuels :
Un fichier java "Graphe" permet de générer un graphe générique qui, selon le choix entre 2 types de graphes sélectionné par l'utilisateur, appellera l'un des deux fichiers "GrapheCrop" ou "GrapheGrass". Ces deux fichiers se chargeront de créer le graphe final avec les spécificités liées au type de culture.

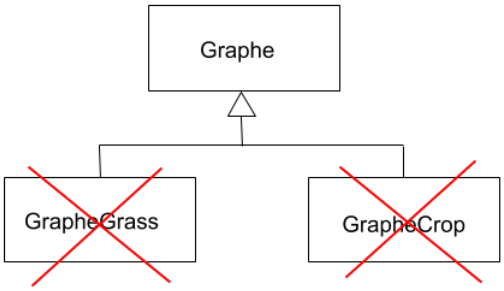


FIGURE 2.10 – Nouvelle architecture de gestion du graphe

A présent, Ces fonctions étant inutiles, nous avons pris la décision de les supprimer, et d'intégrer notre code dans le fichier "Graphe", exceptées les fonctions de lecture qui seront implémentées dans le fichier "InterfaceRésultats" (fichier regroupant toutes les fonctions liées à l'onglet Résultats de FarmSim).Ces fonctions de lecture seront rattachées à l'action du bouton "Chargement des données".

Il a également modifier le chemin d'accès au fichier CSV en entrée : il prenait jusque là directement le nom de la ferme test téléchargée, mais doit à présent s'adapter pour n'importe quelle ferme que l'utilisateur souhaiterait charger.

```
FileReader dirOut = new FileReader(SystemUtils.USER_DIR+File.separator+"xml"+File.separator+farm.getName()+
                    File.separator+"Outputs"+File.separator+farm.getName()+"_general.csv");
```

FIGURE 2.11 – Capture d'écran du nouveau chemin d'accès aux fermes CSV

Ce chemin utilise l'architecture identique entre les fermes : elle prend le nom de la ferme, ouvre son dossier correspondant et cherche le fichier ayant le nom de la ferme puis l'extension "general.csv".

2.2.3 Amélioration de l'interface

Après avoir un graphique fonctionnel intégré dans FarmSim, il fallait résoudre les bugs de celui-ci dans l'interface des résultats :

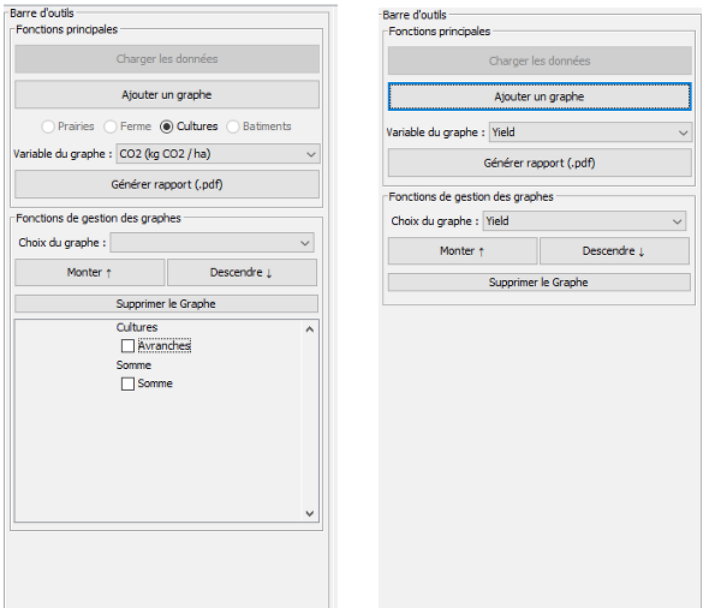


FIGURE 2.12 – Différences entre l'ancienne (à droite) et la nouvelle (à gauche) version de l'interface Résultats

Les boutons" Supprimer Graphe" et "Monter/ Descendre" ne marchaient pas, les boutons " Prairie / Ferme / Culture / Bâtiments" étaient devenus obsolètes.La combobox de permettant de sélectionner la variable du graphe ne fonctionnait pas, et celle de sélection du graphe n'affichait rien. Vu que les graphiques ne s'affichaient pas, le bouton "Générer PDF" exportait un fichier vide. Enfin, les Cultures Avranches et Somme ne faisaient référence à rien, leurs fonctions étaient non abouties.

Les principes de "Supprimer Graphe", "Monter / Descendre" et "Choix du Graphe" étant correct, nous avons juste corrigé leurs fonctions afin qu'ils s'adaptent à notre nouvelle architecture et à notre nouveau code. Ils ont donc été relativement faciles à corriger.
En ce qui concerne le choix de variable du graphe, nous l'avons entièrement codé à nouveau pour qu'il s'accomode de notre nouveau fonctionnement, et qu'il soit à même d'afficher toutes les variables de notre ferme (grâce à la matrice "header").
Pour la génération de PDF, qui affiche à présent nos graphes créés, nous avons simplement supprimé un bouton à cocher qui n'apportait strictement rien au logiciel.
Enfin, nous avons supprimé les boutons " Prairie / Ferme / Culture / Bâtiments" qui n'est plus utile aujourd'hui ainsi que la section Cultures Avranches et Somme qui était beaucoup

trop obscure et inachevée pour être conservée.
En résumé, nous avons épuré notre interface et rendu chaque bouton / fonction présent(e) sur notre onglet Résultats fonctionnel, ce qui rend l'expérience utilisateur bien plus agréable et pratique.

Une fois notre interface corrigée pleinement exploitable,c nous devions ajouter un module d'amélioration du graphe : afficher les types de culture de notre parcelle au cours du temps, sous la forme d'un code couleur en fond avec le type de culture annoté par exemple. Elle permettait d'afficher à l'utilisateur un nouveau type d'information de manière claire et visuelle.

Au cours d'une réunion, nous avons décidé conjointement avec notre tuteur d'implémenter une nouvelle fonctionnalité. Actuellement,la colonne n1 de notre fichier CSV est tout le temps identique, elle correspond à une seule parcelle. A terme, notre fichier de ferme contiendra plusieurs parcelles, donc plusieurs jeu de données, plusieurs courbes, et une gestion assez différente. Plusieurs questions se sont donc posées, sur l'affichage, les avantages et inconvénients de chacune de nos solutions, et la réussite.
Soit :
- nous proposions un graphique séparé pour chaque parcelle, ce qui compliquait leur gestion propre, rendait moins claire leur comparaison, mais simplifiait l'affichage de nos types de culture
- nous affichions un unique graphique pour notre ferme avec des courbes superposées, ce qui facilitait leur interprétation, mais rendait l'affichage moins visible en le surchargeant d'informations.

Après concertation, nous avons privilégié la seconde solution car la priorité du logiciel est de donner la possibilité à l'utilisateur d'interpréter les courbes qui s'affichent, superposer les courbes nous parait donc plus intéressant, et donc prioritaire sur l'affichage des types de culture. Nous prenons donc la décision de ne plus afficher nos changements de culture sous la forme de couleurs en fond, puisque ce serait impossible de superposer les couleurs en fonction des parcelles. A la place, nous allons créer des jalons : des traits verticaux qui, à une date donnée, écrivent pour une parcelle le changement de culture (Exemple : Parcelle "Chaillol" : Maïs -> Blé). Ainsi, on peut visualiser de façon plus claire les dates de changement de culture.

Pour implanter notre nouvelle solution, nous avons dû modifier notre matrice pour qu'elle prenne en compte nos changements de parcelles ou de culture (Cf. partie "Création du graphe", figures 2.5 et 2.7). A partir de notre matrice modifiée, qui associe à présent des identifiants de parcelle/ culture à un nom, nous devons gérer l'affichage de nos jalons.

```
// Creation des jalons
if(cultureCour != culturePrec) {
  int yearChange = (int) matrix[i][2];
  int doyChange = (int) matrix[i][3];
  Calendar cal = Calendar.getInstance();
  cal.set(Calendar.DAY_OF_YEAR, doyChange);
  int dayChange = cal.get(Calendar.DATE);
  int monthChange = cal.get(Calendar.MONTH)+1;

  TimeSeriesDataItem item = timeSeriesCour.getDataItem(timeSeriesCour.getItemCount()-1);

  double x = item.getPeriod().getFirstMillisecond();
  double y = item.getValue().doubleValue();

  RegularTimePeriod dateChange = new Day(dayChange, monthChange, yearChange);
  ValueMarker jalon = new ValueMarker(dateChange.getMiddleMillisecond());

  XYTextAnnotation labelJalon = new XYTextAnnotation(parcelleLinkId.get(parcelleCour)+" : "+cultureLinkId.get(culturePrec)+
                                                     "->"+cultureLinkId.get(cultureCour), x, y);
  labelJalon.setRotationAngle(-Math.PI/2);
  labelJalon.setFont(new Font("SanSerif", Font.PLAIN,15));
  labelJalon.setRotationAnchor(TextAnchor.BOTTOM_LEFT);
  labelJalon.setTextAnchor(TextAnchor.BOTTOM_LEFT);

  jalon.setPaint(Color.black);
  jalon.setStroke(new BasicStroke(1F));

  listeAnnotation.add(labelJalon);

  listeJalon.add(jalon);
```

FIGURE 2.13 – Capture d'écran du code de création des jalons

Pour cela, nous devons récupérer le moment où il y a un changement de culture lors du parcours de notre matrice, convertir cette date en " Calendar" (comme expliqué précédemment, notre date doit être convertie afin d'être prise en compte dans JFreeChart), puis nous créons notre jalon, de type "ValueMarker".
"Value Marker" est un type qui crée un trait à la valeur souhaitée(horizontal ou vertical), et qui affiche le texte désiré.
Cependant, le texte d'un ValueMarker est "figé" : c'est à dire qu'il ne peut s'afficher qu'à l'horizontale, dans une certaine taille, en haut de notre graphique. Vu la taille du texte que l'on souhaite insérer (Parcelle "Chaillol" : Maïs -> Blé) et le nombres de jalons créés, nos textes se chevauchent et deviennent illisibles.

Pour pallier à ce problème, nous avons dû créer un nouveau type de texte plus souple, qui serait rattaché à notre jalon. Nous utlisons donc un type "XYTextAnnotation", qui permet de créer du texte, que l'on "accoche à une valeur" (ici la date de notre jalon), que l'on peut personnaliser. Angle, taille, couleur, position... Grâce à ça, nous pouvons afficher notre texte de façon horizontale, notre affichage global devenant plus clair et plus lisible.
Voici le rendu final :

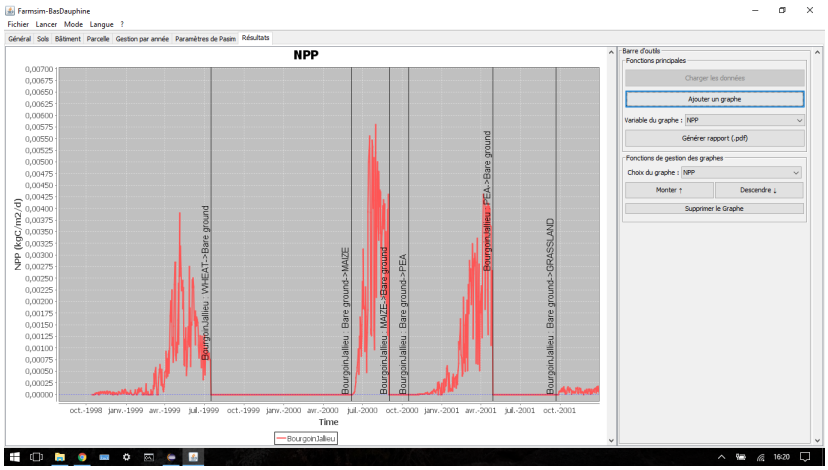


FIGURE 2.14 – Capture d'écran de l'affichage d'une parcelle avec les jalons

Chapitre 3

Résultats et Discussion

Dans cette section, nous présentons le rendu final, les différentes fonctionnalités qui composent notre solution et comment l'utiliser, mais nous abordons les limites de nos résultats ainsi que les perspectives d'évloution de ce projet dans le temps.

3.1 Résultat final

L'onglet Résultats, sur lequel nous avons travaillé tout au long de notre projet, se présente aujourd'hui comme ceci :

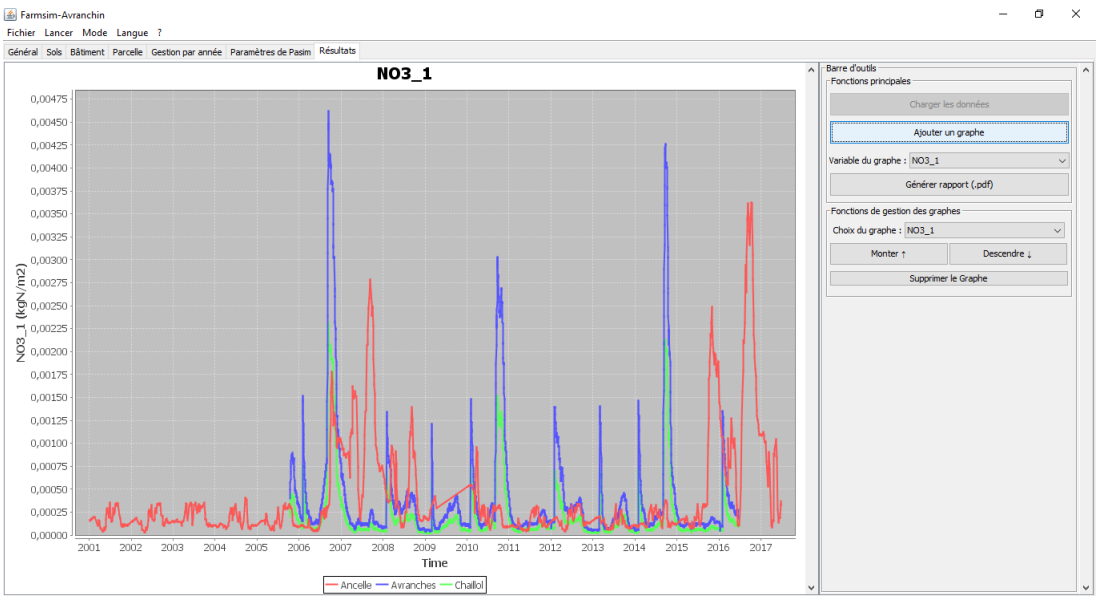


FIGURE 3.1 – Capture d'écran d'une création de graphique

Pour pouvoir créer un graphique, il faut dans un premier temps appuyer sur le bouton

"Chargement des données". Le clic entraînera un parcours de la ferme, puis le stockage des différentes données dans des matrices adaptées. Un fois le chargement effectué, le bouton deviendra grisé : il ne pourra plus être cliqué. Un autre bouton deviendra actif à la place, le bouton "Ajouter un graphe".
Avant de cliquer sur ce bouton, l'utilisateur aura la possibilité au préalable de choisir quelle variable afficher sur son graphique : une liste déroulante sera mise à sa disposition pour choisir parmi toutes les variables disponibles sur cette ferme. Lorsqu'il aura effectué son choix en cliquant sur la variable souhaitée, il pourra ensuite cliquer sur le bouton " Ajouter Graphe", qui va donc prendre en paramètres notre matrice de données ainsi que le variable choisie, puis va construire et afficher le graphique pour donner le résultat que l'on peut voir sur la figure 3.1, ou bien la 3.2 (Nous avons pris le choix de décomposer l'affichage en 2 pour plus de clarté).

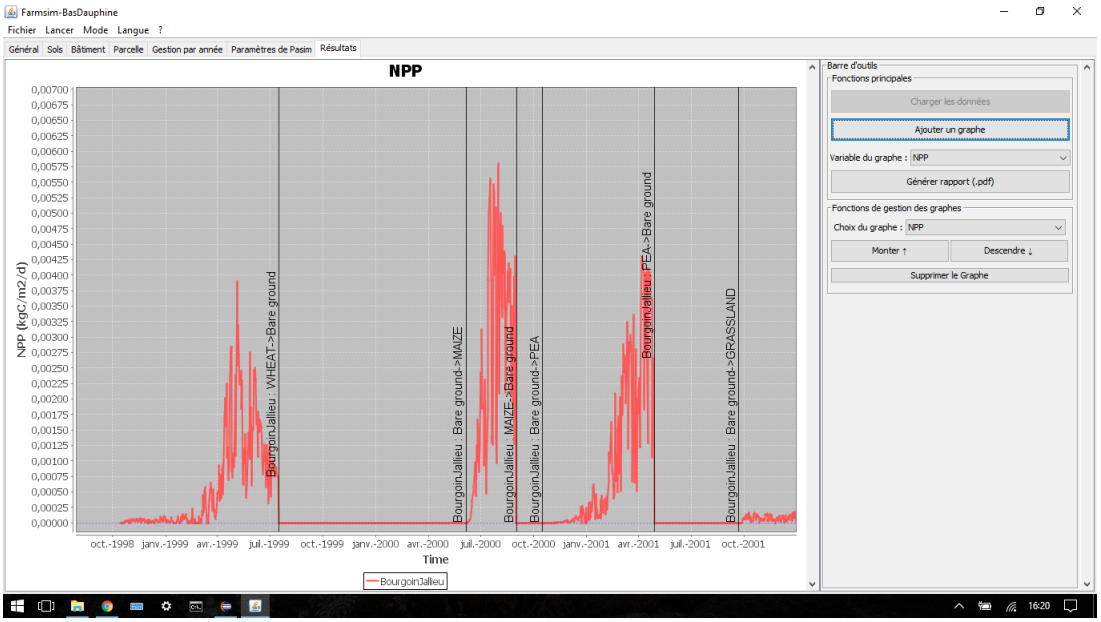


FIGURE 3.2 – Capture d'écran d'un résultat avec affichage de jalon

On peut répéter l'opération à notre guise pour afficher autant de graphiques que souhaité.

Afin de pouvoir mieux les réorganiser à sa convenance, l'utilisateur peut sélectionner un de ces graphiques par l'intermédiaire d'une seconde liste déroulante (elle s'affiche dans l'ordre actuel des graphiques), ce qui mettra ce graphique dans une couleur différente, puis pourra cliquer sur "monter" ou "descendre" et ainsi déplacer son graphique.

Une fois les graphiques bien ordonnés, l'utilisateur a la possibilité d'exporter tous nos graphiques dans un fichier de format PDF : il suffit simplement de cliquer sur le bouton "générer rapport", qui enregistrera tous les graphiques à la suite et les exportera en format PDF, puis affichera le fichier en question pour montrer le rendu.

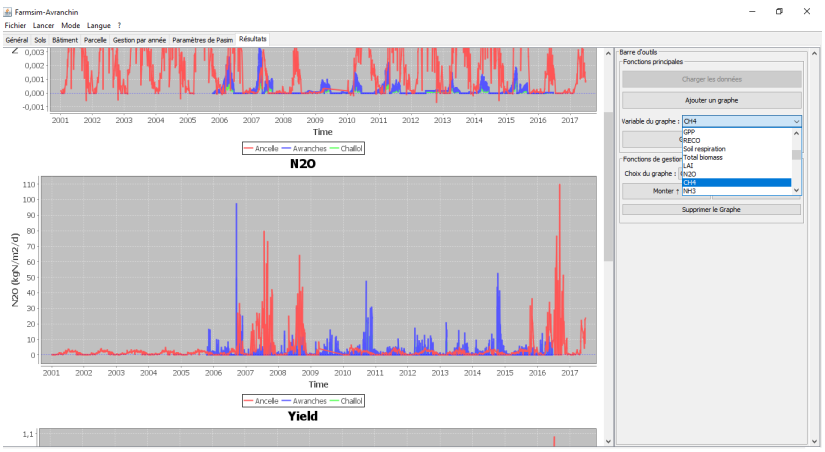


FIGURE 3.3 – Capture d'écran de plusieurs graphes

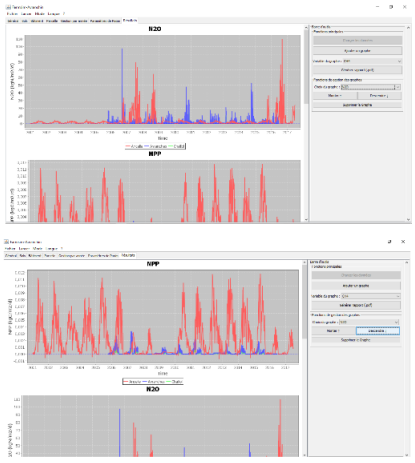


FIGURE 3.4 – Capture d'écran de la descente d'un graphe

3.2 Limites et améliorations possibles

Nous sommes globalement satisfaites de notre résultat : par rapport à nos objectifs initiaux, nous avons été en mesure de rendre une interface fonctionnelle, améliorée et plutôt claire. Nous avons cependant conscience que notre solution est loin d'être optimale, que certains aspects sont plus liés à de la "bidouille" qu'à une véritable solution stable.

Par exemple, notre système d'affichage de changement de culture est pour le moins approximatif, et grandement perfectible. Accrocher une annotation à un marqueur est une solution qui dépend de trop de variables et risque plus facilement de ne plus fonctionner, mais c'est la seule solution que nous avons trouvé dans la documentation de JFreeChart actuellement.
De plus, lorsque l'on combine l'affichage de plusieurs parcelles en simultané et nos jalons pour chaque parcelle, nous avons une surcharge d'informations qui rend notre graphique très peu lisible, ce qui est très problématique puisque c'est le coeur même de cette interface. Pour éviter ce problème, nous pourrions imaginer un second graphique en-dessous (JFree-Chhart permet cela :en créant un second DataSet par exemple) qui se concentrerait sur ces changement de culture. Nous aurions donc une information certes décomposée, mais suffisamment proche pour permettre d'interpréter les données, et nos graphiques seraient plus clair.

Une autre limite de notre affichage actuel est que nous nous affichons un unique type de graphique : une courbe. Mai en fonction des différents besoins d'analyse de l'utilisateur il serait peut-être judicieux de proposer différents types de graphiques, avec n'importe quel type de variable pour vraiment proposer une expérience ultra personnalisée, comme des camemberts pour comparer la part de chaque culture dans une parcelle, ou n'importe quelle problématique que l'on pourrait etre à même de se poser. Cette partie pourrait représenter une grande perspective d'évolution à l'avenir.

Une fonctionnalité qui nous semblerait pertinente et intéressante pour la suite serait de proposer l'ajout de commentaire ou de texte autour de ces graphiques. Ainsi, la personne pourrait noter ses travaux, ses observations, ses hypothèses ou encore ses conclusions, et lors de l'export en PDF nous n'aurions plus uniquement une succession de graphiques mais un rapport plus complet sur les données.

Conclusion

Nous avons été mandatées par l'INRA et en particulier le service de l'UREP pour effectuer un travail lié à l'Interface Homme/Machine et à l'expérience utilisateur.
Pour ce projet, nous avons dû améliorer et adapter toute l'interface d'affichage de résultats du logiciel FarmSim, l'interface actuelle étant devenue inutilisable car obsolète.
Nous avons dû recréer de toute pièce un nouveau code Java en utilisant la bibliothèque JFreeChart afin de permettre l'affichage d'un graphique, puis incorporer ce travail dans le logiciel FarmSim. Une fois ceci effectué, nous avons eu pour mission de nettoyer le code du logiciel et corriger les bugs entravant une utilisation correcte de FarmSim, et enfin améliorer son utilisation en ajoutant de noveaux modules et affichage.
Ce projet a vraiment été pour nous l'occasion de nous épanouir sur un sujet qui nous tenait à coeur, à la fois sur le plan technique et le plan personnel :
- Technique pour la dimension d'Ergonomie, d'Affichage et de Java, qui sont des technologies qui nous intéressent et que nous souhaitions découvrir et développer pour notre apprentissage
- Personnel pour les enjeux que ce projet représente, l'écologie et la protection de l'environnement sont des notions qui nous sont sensibles, et nous avions à coeur de travailler sur un projet qui nous semblait utile au bien commun et à l'évolution des recherches de ce domaine.
Nous sommes plutôt satisfaites de ce que nous avons produit, car malgré un début de projet assez compliqué et du retard accumulé suite à nos différents blocages techniques, nous avons su rebondir et passer beaucoup de temps à réfléchir dessus pour satisfaire aux demandes de notre tuteur, qui lui-même était disposé à tout mettre en oeuvre pour nous aider à avancer.
Tous ces facteurs nous ont permis de remplir les objectifs que nous nous étions fixés

Sitiographie

INRA - http ://institut.inra.fr/

UREP - https ://www1.clermont.inra.fr/urep/

JFREE - http ://www.jfree.org/forum/

STACKOVERFLOW - https ://stackoverflow.com/

OPENCLASSROMM - https ://openclassrooms.com/courses/apprenez-a-programmer-en-java

DEVELOPPEZ.COM - https ://java.developpez.com/

DOC JAVA - https ://docs.oracle.com/javase/7/docs/api/

MATHEMATEX - http ://forum.mathematex.net/

SHARELATEX - https ://fr.sharelatex.com/learn/MainPage

Glossaire

FarmSim *Farm Simulation model.* Modèle de ferme permettant le calcule de bilan de GES.

GES *Gaz à Effet de Serre.* Ensemble des composants gazeux qui absorbent le rayonnement infrarouge émis par la surface terrestre, contribuant ainsi à l'effet de serre. Les principaux GES sont la vapeur d'eau, le dioxyde de carbone, le méthane, le protoxyde d'azote et l'ozone.

IDE *Integrated Development Environment.* Programme regroupant un ensemble d'outils pour le développement de logiciels.

INRA *Institut National de la Recherche en Agronomie.*

ISIMA *Institut Supérieur d'Informatique, de Modélisation et leurs Applications.*

PaSim *Pasture Simulation model.* Modèle mécaniste biogéochimique de simulation d'un écosystème prairial géré simulant, à l'échelle d'une parcelle, les flux de carbone, d'azote, d'eau et d'énergie à l'interface entre le sol, la végétation, les animaux et l'atmosphère.

UREP *Unité de Recherche dans l'Ecosystème Prairial.*

Annexes

FIGURE 5 – Diagramme de Gantt prévisionnel

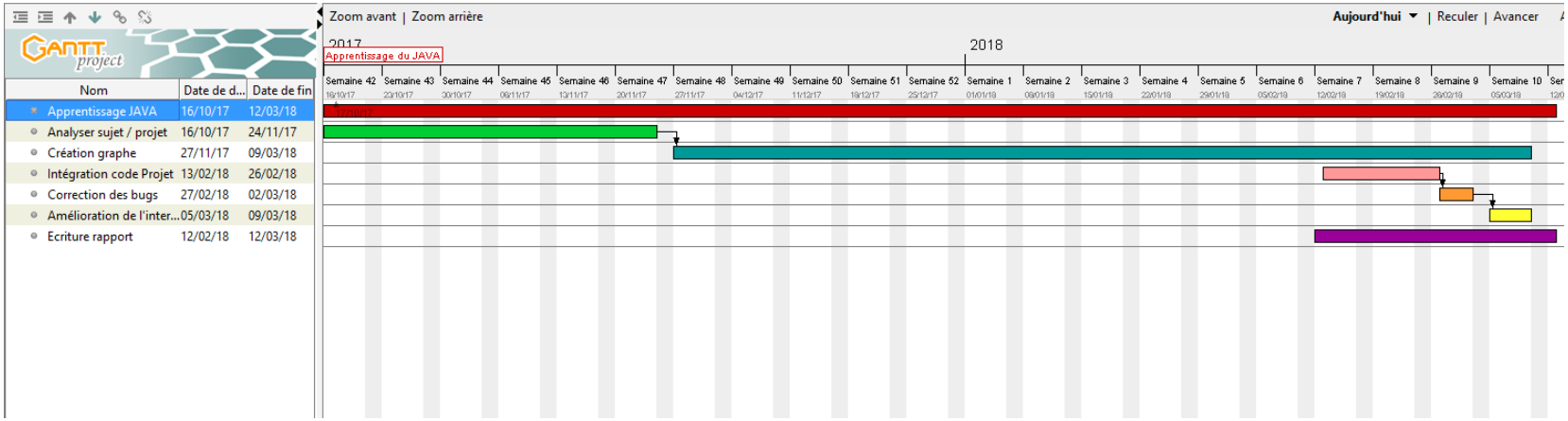


FIGURE 6 – Diagramme de Gantt réel