Université d'Orléans
Département informatique

RAPPORT
Projet Informatique

Réalisé par :

- Tennessy KOLUBAKO
- Khaled MAAROUFI
- Jimmy PADONOU LOUEMBET
- Adrielle YOUGANG

Contenu

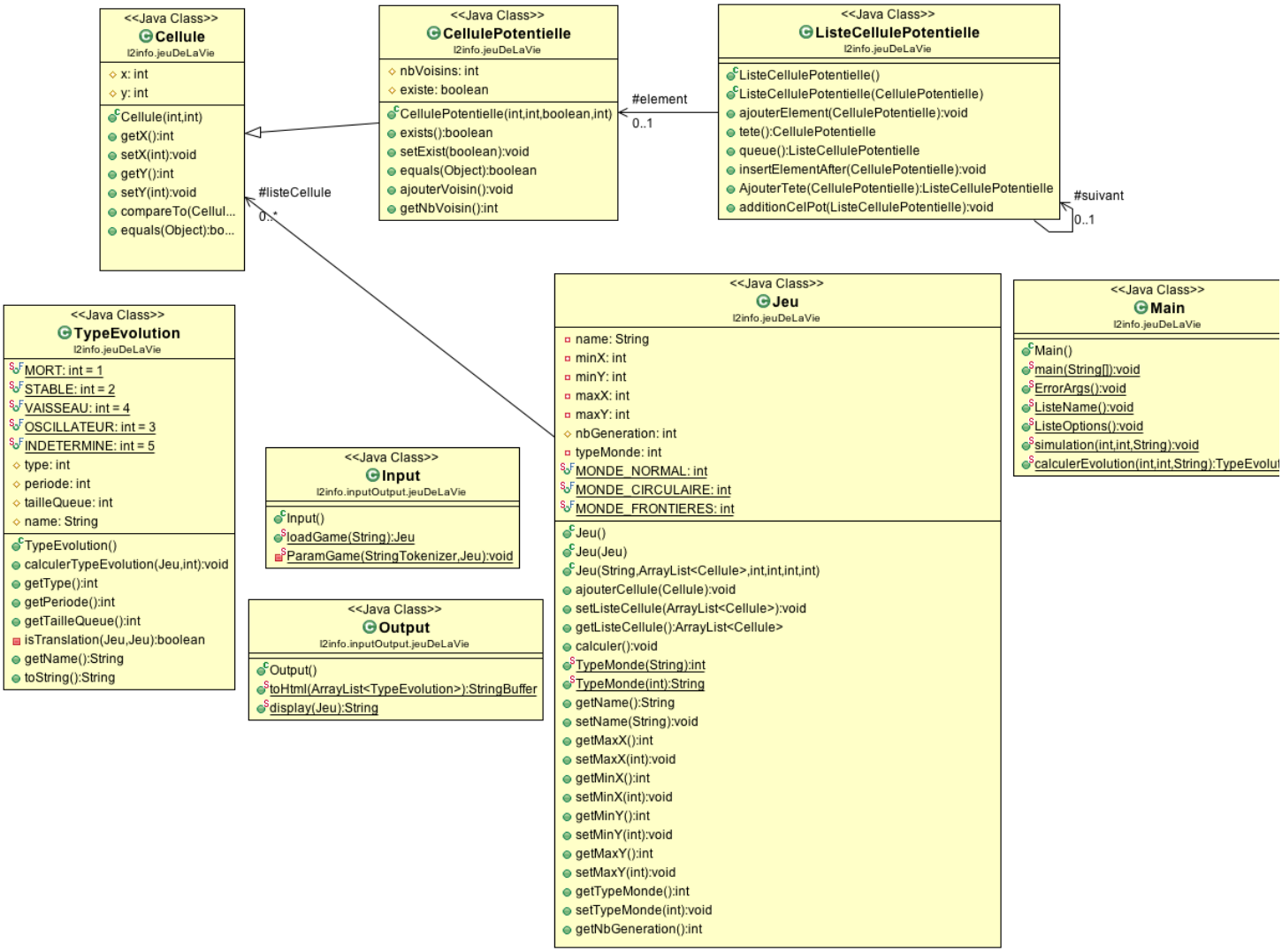
Le 24 avril 2013

L'objectif du projet est de réaliser une application permettant de lire des fichiers au format « .lif » et de simuler le jeu de la vie décrit par ces fichiers tout en respectant les règles fondamentales décrites dans le sujet. Des éléments spécifiques tels que la période et la taille de la queue sont aussi à déterminer pour chaque jeu simulé.

I. Architecture de l'application & Caractéristiques:

I.1. Architecture

Voici le diagramme de classes résumant les méthodes utilisées :



8 classes nous ont été utiles pour réaliser le travail souhaité :

« Cellule.java » contient des coordonnées , en d'autres mots elle est identique à un point (une abscisse et une ordonnée). Une instance de cette classe représente une cellule du jeu.

« CellulePotentielle.java » hérite de la classe précédente et compte des attributs en plus (ex : nbvoisin ...) qui est une cellule morte susceptible de naître dans la génération à venir. Cette classe est utilisée lors du calcule d'une nouvelle génération. Nous avons fait le choix de créer cette classe afin de ne pas avoir a rajouter des attributs à la classe Cellule qui ne seraient utiliser que lors de ce calcule précis.

« ListeCellulePotentielle.java » est semblable à une liste chainée du point de vue fonctionnement. Elle contient des Maillons de CellulesPotentielles.

« Jeu.java » est la classe contenant tout ce qui rentre en compte dans la generation d'un Jeu de la vie, ainsi que le calcule des nouvelles générations.

« Main.java » fait tous les tests avant de lancer le jeu sur les arguments en paramètres.

« TypeEvolution.java » contient le nécessaire au calcule du type d'évolution d'un jeu (Vaisseau, stable, etc…)

« Input.java » s'occupe de tout ce qui est lecture des fichiers lif.

« Output.java » s'occupe des differents affichages utilisés (Affichage de la disposition des Cellules d'un Jeu, generation du code HTML, etc…) .

I.2. Caractéristiques

Cependant, chacune des classes à une classe de test réalisées via J-Unit dans un package différent. Nous avons choisis, dans un soucie de clarté, de séparer notre projet en trois packages : Le premier contenant les classes du jeu, le second les classes d'Input/Output et le dernier contenant les classes de Test. La structure de données choisies parmi celles imposées est l' ArrayList.

Elle répond au jour d'aujourd'hui à la totalité du cahier des charges qui nous a été fourni, y compris la simulation des mondes frontière et circulaire. L'ensemble des options concernant l'application sont disponibles en exécutant la commande suivante dans un terminal: `#java -jar JeuDeLaVie.jar -h` .

Pour exécuter au moins la simulation d'un fichier il est conseillé de placer le fichier dans le même répertoire que l'archive en « .jar » ou un sous dossier du répertoire(ex : Si le fichier sont dans le sous répertoire 's_reg' il suffit d' écrire 's_reg/nom-fichier.lif').

II. Difficultés rencontrées & gestion

II.1. Difficultés rencontrées

Pour éviter tous conflits de données et avoir toujours un avancement identique avec chaque membre du groupe nous avons choisi « Github » comme gestionnaire de dépôts.

De nombreux problèmes ont étés rencontrés dans la réalisation du projet :

-Les tests unitaires : Chacune des classes du jeu ont été réalisées avant les tests et pour cela nous avons eu du mal à gérer la visibilité de certaines méthodes car celles-ci étaient obligatoires à la bonne couverture de l'application . Pour régler ce problème nous avons donc choisi de laisser leur visibilité en public .

-L' optimisation : Dans un soucie d'optimisation et d'efficacité, il nous est arrivé de devoir revoir nos algorithmes car ceux-ci n'étaient pas aussi efficace que nous le pensions. C'est par exemple le cas de la méthode calculer, qui a subit de nombreuses réécriture.

-Gestion des cellules vivantes : Avant de pouvoir utiliser la structure de données imposé il nous a fallu dans un premier utiliser des tableaux de cellules (pour des tailles finies de l'échiquier) afin de nous donner un aperçu et enfin l' implémenter.

-Un autre problème que nous avons rencontré dans les débuts de ce projet est l'impossibilité de pouvoir ajouter des objet dans une ArrayList tout en la parcourant. C'est pourquoi nous avons décidé de créer la classe ListeCellulePotentielle, qui nous permet ainsi d'ajouter des éléments à un endroit précis tout parcourant une instance de celle-ci.

II.2. Gestion des taches

Quant à la répartition des tâches au sein du groupe, celle-ci à prit forme de la façon suivante :

- *KOLUBAKO Tennessy*
  - Écriture des classes Jeu, CellulePotentielle, ListeCellulePotentielle, Input, Output, TypeEvolution
  - Tests unitaires J-Unit
  - *Réorganisation du code*
- *MAAROUFI Khaled*
  - Spécifications
- *PADONOU LOUEMBET Jimmy*
  - Écritures des classes Main et Cellule, méthodes de lecture
  - Tests unitaires J-Unit
- *YOUGANG Adrielle*
  - Spécifications

De plus chacun a été aidé par les autres membres du groupe dans l'implémentation de certaines fonctions quand celui-ci rencontrait des difficultés.

III. Résultats des expérimentations

Ci-dessous un tableau comprenant tous les résultats de chaque fichier

	MONDE FRONTIÈRE			*MONDE CIRCULAIRE*			*MONDE NORMAL*		
	Période	*Taille de la queue*	*Évolution*	*Période*	*Taille de la queue*	*Évolution*	*Période*	*Taille de la queue*	*Évolution*
Fichier.lif	68	68	O	1	69	M	1	23	M
Per2.lif	2	2	O	6	6	O	2	2	O
Per3.lif	93	93	O	152	152	O	18	18	O
Vaiss.lif	8	8	O	1	8	M	4	4	V
Sta456.lif	1	1	S	57	57	O	1	1	S

Légende :

- M= mort ; V= vaisseau ; O= oscillateur ; S= stable