Rapport Jeu de la Vie LIFAPOO

Diego GALFRÉ
Delshad KADDO
Licence 3 Informatique

1. Introduction

Le Jeu de la Vie, créé par John Conway en 1970, est une simulation qui se déroule sur une grille 2D où chaque cellule peut être morte ou vivante. Les cellules évoluent selon des règles simples, et cette dynamique produit des motifs de plus en plus complexes.
Le Jeu de la Vie est aussi un concept qui permet d'hypothétiser comment la vie a pu émerger sur Terre, en montrant comment des processus simples peuvent conduire à des comportements complexes et évolutifs, comme dans les systèmes biologiques.

Les règles sont les suivantes :

1. Une cellule vivante avec moins de 2 voisines vivantes meurt (sous-population).
2. Une cellule vivante avec 2 ou 3 voisines vivantes reste vivante.
3. Une cellule vivante avec plus de 3 voisines vivantes meurt (sur-population).
4. Une cellule morte avec exactement 3 voisines vivantes devient vivante (reproduction).

Projet

Ce projet a été développé dans le cadre du cours LIFAPOO à l'Université Claude Bernard Lyon 1. Il a pour objectif de mettre en pratique les concepts de la programmation orientée objet à travers la réalisation d'une simulation du Jeu de la Vie. Ce projet nous a permis d'appliquer des notions fondamentales telles que la gestion des classes, des objets, ainsi que des interactions entre différents composants d'un programme dans un environnement graphique.

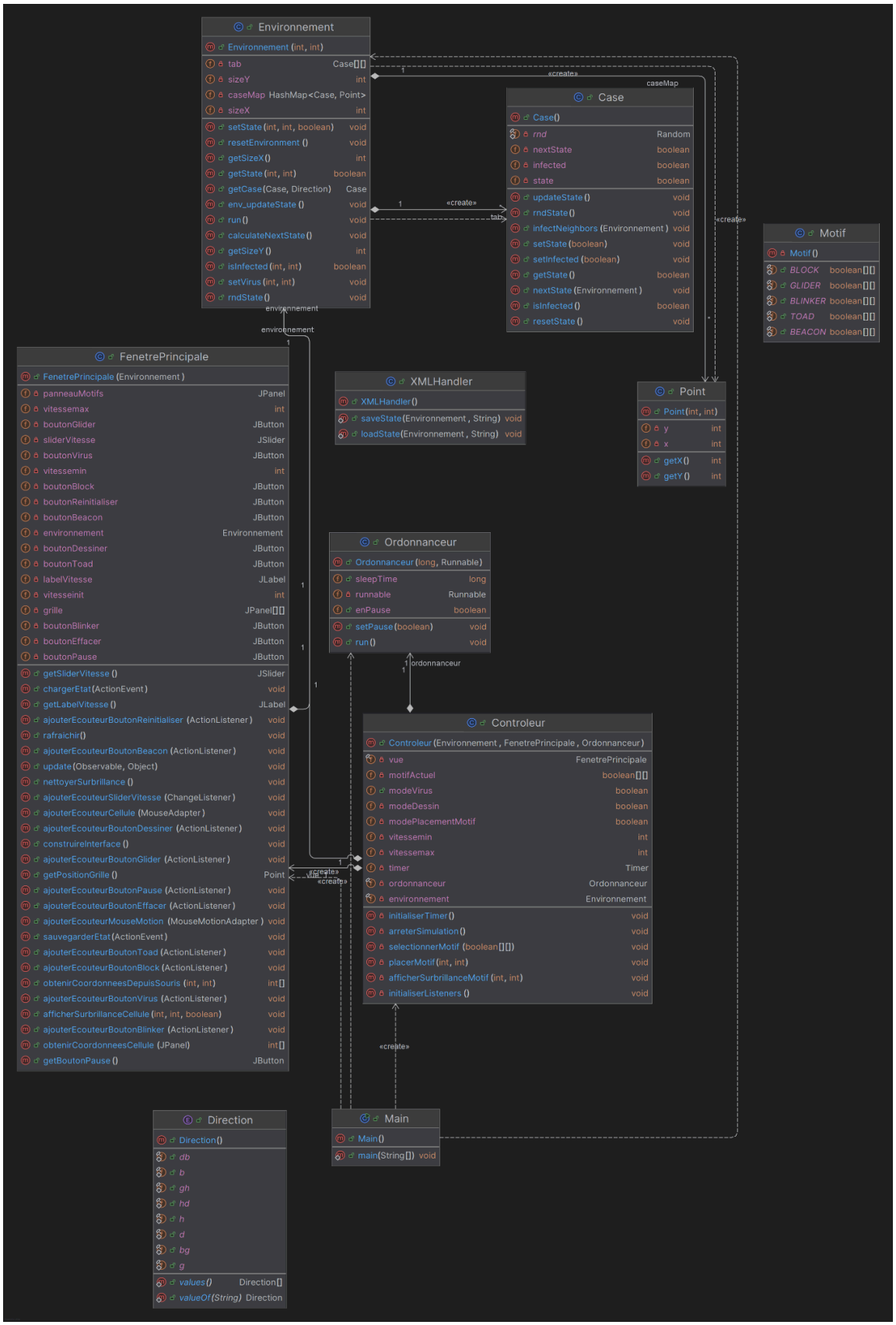
Coder ce jeu en Java est intéressant car il permet de bénéficier de la programmation orientée objet pour organiser le code, d'outils comme Swing pour créer des interfaces graphiques.

Le projet a débuté avec un code de base qui générait simplement une grille affichant des états aléatoires à chaque génération. À partir de ce point de départ, l'objectif était d'enrichir cette simulation en ajoutant des fonctionnalités supplémentaires, l'implémentation des règles, la possibilité d'interagir avec la grille, et l'ajout de mécanismes de contrôle de la simulation, comme la gestion de la vitesse, la pause, et l'activation de différents modes de fonctionnement.

Architecture MVC:

Comme ce qui a été demandé dans l'énoncé de ce projet, et pour compléter l'architecture MVC, on a dû ajouter une classe contrôleur dans le répertoire vue_controleur. Cette classe joue le rôle d'intermédiaire, coordonnant la communication entre le modèle (Environnement et Ordonnanceur .. etc) et la vue (FenetrePrincipale.java), ce qui sera détaillé plus précisément lors de la présentation explicite de chaque classe ci-dessous.

2. Diagramme UML (Diagramme de Classe aussi dispo dans GOL/src/data)



3. Description des Classes Principales

3.1. Classe Case

La classe Case représente une cellule individuelle dans la grille du jeu de la vie. Cette classe est utilisée pour encapsuler les données relatives à chaque cellule de la grille.

-Rôles principaux:

- Stocker l'état de la cellule (true ou false), indiquant si la cellule est vivante ou morte.
- Gérer l'interaction avec l'état de la cellule (changement d'état en fonction des règles du Jeu de la Vie).

-Exemple de méthode :

- *nextState(Environnement env)* : Calcule l'état suivant de la cellule en fonction des règles du Jeu de la Vie. La méthode compte le nombre de voisins vivants de la cellule et met à jour son état selon les règles suivantes :

3.2. Classe Environnement

La classe Environnement représente l'environnement global du Jeu de la Vie, gérant l'état de la grille de cellules. Elle est responsable du calcul de l'état suivant des cellules et de leur mise à jour à chaque itération du jeu.

-Rôles principaux:

- Gestion de la grille : La classe maintient une grille de cellules, où chaque cellule est représentée par un état binaire (true ou false).
- Calcul de l'état suivant : La logique de calcul de l'état suivant de chaque cellule est implémentée ici, en respectant les règles du Jeu de la Vie (nombre de voisins vivants, etc.).
- Modification de l'état d'une cellule : Elle fournit des méthodes pour modifier l'état des cellules.

-Exemples de méthodes :

- *setState(x, y, state)* : Modifie l'état d'une cellule.
- *calculateNextState()* : Calcule l'état suivant de toutes les cellules, basé sur leurs voisins.

3.3. Classe FenetrePrincipale

La classe FenetrePrincipale est l'interface graphique principale du projet. Elle gère l'affichage de la grille, ainsi que les interactions avec l'utilisateur via des composants visuels tels que des boutons, ou l'ajout de cellules par l'utilisateur.

-Rôles principaux:

- La vue affiche la grille de cellules et met à jour le visuel de la grille en fonction de l'état de ses cellules (true ou false).
- Gestion des interactions : capture les événements d'entrée de l'utilisateur (clics de souris, mouvements, etc.) pour permettre des actions (modification de l'état des cellules...)
- Contrôles de simulation : Avec des boutons pour démarrer, mettre en pause ou réinitialiser la simulation, ainsi que pour ajuster la vitesse de simulation (Détail dans la partie 4).

3.4. Classe Controleur

La classe Controleur fait le lien entre la vue et le modèle. Elle gère les actions de l'utilisateur (clics, boutons, etc.) et met à jour l'état du modèle en conséquence.

-Rôles principaux:

- La classe capte les événements de l'utilisateur provenant de la vue (clics sur la grille, boutons).

-Exemples de méthodes :

- *gererClicOuDrag(MouseEvent e)* : Gère le clic ou le glisser-déposer sur la grille, modifiant l'état des cellules en fonction de l'action.

- *selectionnerMotif(boolean[][] motif)* : Sélectionne un motif prédéfini à placer sur la grille.
- *arreterSimulation()* : Arrête l'ordonnanceur et met en pause la simulation.

3.5. Classe Ordonnanceur

La classe Ordonnanceur gère l'exécution du jeu en contrôlant le rythme de la simulation via un mécanisme de temporisation.

-Rôles principaux:

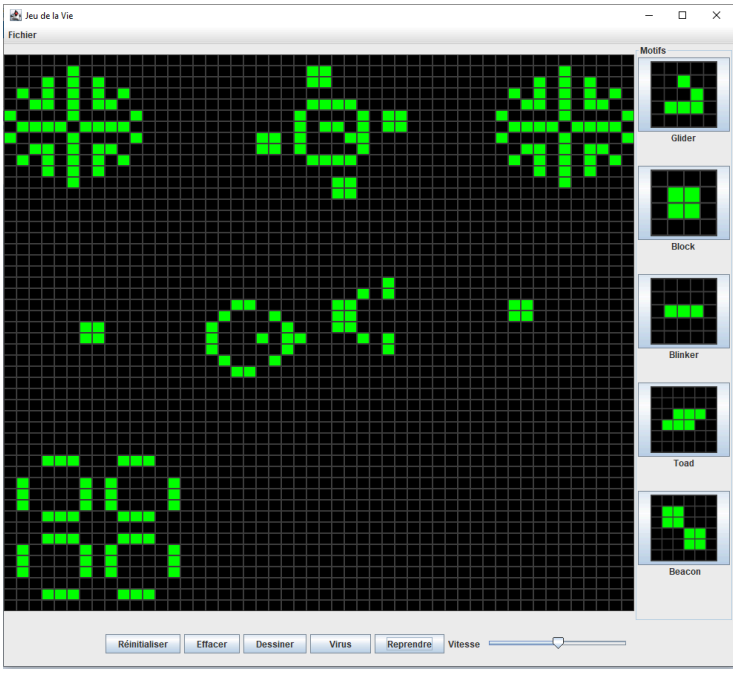
- L'ordonnanceur est responsable du calcul de l'état suivant et de la mise à jour de l'affichage à intervalles réguliers.
- Pause et reprise : Il permet de mettre en pause la simulation et de la reprendre sans perdre l'état actuel.

-Exemples de méthodes :

- *setPause(boolean pause)* : Met la simulation en pause ou la reprend.
- *run()* : Exécute la simulation en boucle, en appelant la méthode du modèle pour calculer et mettre à jour l'état des cellules à chaque itération.

4. Description des Extensions

Dans le cadre de ce projet, plusieurs extensions ont été ajoutées pour rendre le Jeu de la Vie plus interactif et fonctionnel. Ces extensions visent à améliorer l'expérience utilisateur, offrir plus de flexibilité dans l'utilisation de l'application et étendre les possibilités de simulation. Voici une description détaillée des principales extensions développées :



4.1. Contrôle du Temps de Simulation (A)

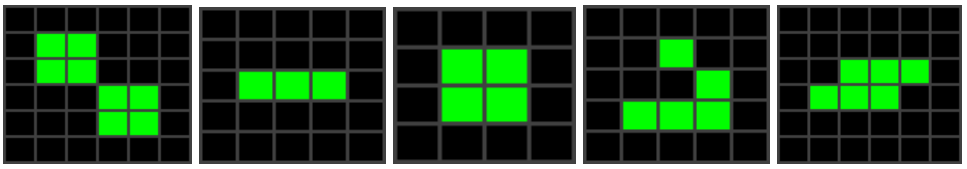
- Objectif : Permettre à l'utilisateur de mettre en pause la simulation et de contrôler la vitesse à laquelle les générations évoluent.
- Description : Un bouton de pause a été ajouté à l'interface, permettant à l'utilisateur d'interrompre temporairement la simulation. Un curseur permet également d'ajuster la vitesse de la simulation, en modifiant le sleeptime entre chaque génération.
- Temps de réalisation estimé : 15% du temps consacré aux extensions.

4.2. Environnement Interactif (A)

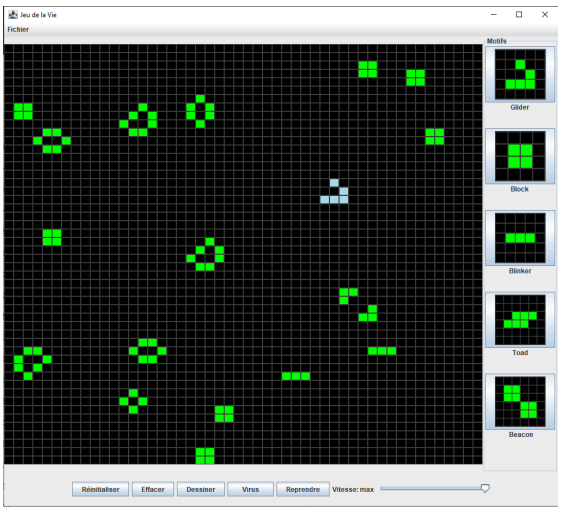
- Objectif : Permettre à l'utilisateur de manipuler directement l'état des cellules de la grille en cliquant dessus et de générer un environnement vide.
- Description : L' utilisateur peut directement cliquer sur les cellules de la grille pour inverser son état. Un bouton permettant de réinitialiser la grille aléatoirement a également été ajouté. Enfin, le bouton “Effacer” permet de réinitialiser l’environnement à un état vide.
- Temps de réalisation estimé : 10% du temps consacré aux extensions.

4.3. Proposer des Motifs Pré-Construits à Placer dans l'Environnement (B)

- Objectif : Offrir la possibilité à l’utilisateur d’ajouter des motifs prédéfinis dans l’environnement, avec une prévisualisation avant placement.
- Description : Plusieurs motifs classiques du Jeu de la Vie (*Glider*, *Block*, *Blinker*, Toad) ont été ajoutés dans une classe Motif. Ces motifs peuvent être placés facilement dans l’environnement. Lorsqu’un motif est sélectionné, il apparaît en surbrillance lorsqu’il est survolé par la souris, permettant à l’utilisateur de le positionner exactement où il le souhaite avant de le fixer.
- Images des motifs :



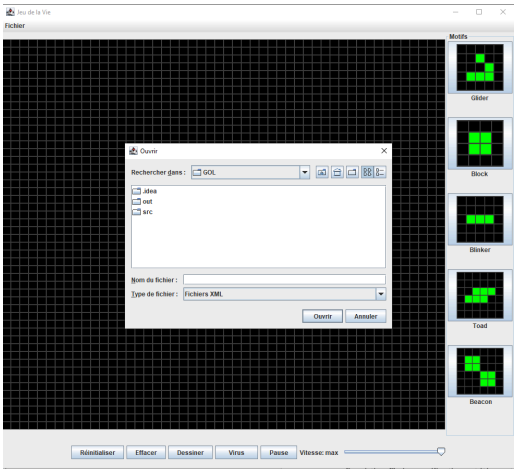
- Surbrillance des motifs avant leur placement (en bleu-gris):



- Temps de réalisation estimé : 25% du temps consacré aux extensions.

4.4. Charger et Sauvegarder l'État des Grilles de Cellules (B)

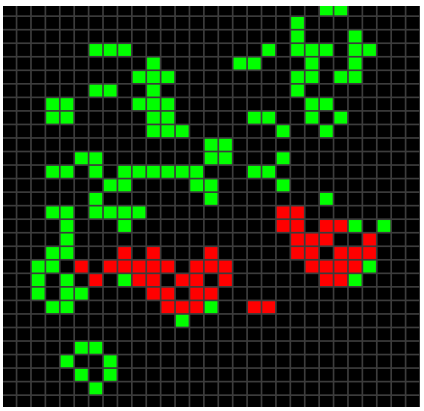
- Objectif : Permettre à l’utilisateur de sauvegarder l’état actuel de la grille et de le recharger plus tard pour reprendre une simulation ou tester différents scénarios.
- Description : La fonctionnalité de sauvegarde et de chargement de l’état de la grille a été ajoutée. L’état de la grille est exporté vers un fichier XML, où chaque cellule et son état (vivant ou mort) sont enregistrés. L’utilisateur peut ensuite charger un fichier XML pour restaurer l’environnement dans son état précédent.



- Temps de réalisation estimé : 20% du temps consacré aux extensions.

4.5. Fonctionnalité Virus

- Objectif : ajouter une dimension supplémentaire au Jeu de la Vie en permettant à certaines cellules d'être "infectées", et de suivre des règles spécifiques.
- Description : En activant le mode "Virus", l'utilisateur peut infecter une cellule en cliquant dessus. Les cellules infectées deviennent rouges et se comportent comme les cellules classiques, mais contaminent les cellules voisines. Ce mode introduit un nouvel aspect au jeu, permettant de simuler des phénomènes comme la propagation d'un virus à travers une population de cellules.
- Temps de réalisation estimé : 20% du temps consacré aux extensions.



4.6. Réglages et Améliorations Diverses

- Objectif : Améliorer l'interaction avec l'application et offrir plus de flexibilité à l'utilisateur.
- Description : Plusieurs améliorations ont été apportées à l'interface pour rendre l'expérience utilisateur plus fluide et intuitive. Parmi ces améliorations, l'ajout du mode dessin permet à l'utilisateur de créer ou de modifier des motifs directement sur la grille en cliquant et en faisant glisser la souris. De plus, un bouton de réinitialisation a été introduit pour permettre à l'utilisateur de rapidement réinitialiser l'environnement à un état de départ aléatoire
- Temps de réalisation estimé : 10% du temps consacré aux extensions.

5. Conclusion

Ce projet du Jeu de la Vie nous a permis d'approfondir nos compétences en programmation orientée objet et en développement Java, notamment avec l'utilisation de Swing pour l'interface graphique. Nous avons appris à structurer un projet en suivant l'architecture MVC. L'implémentation des fonctionnalités nous a permis de mieux maîtriser la gestion des interactions utilisateur, la mise à jour dynamique de l'interface et la gestion des états au sein d'un programme interactif. En somme, ce projet a enrichi notre expérience en développement d'applications Java, en gestion d'interfaces graphiques et en conception de simulations dynamiques.