Binauld Pierre
Roger Quentin

Jeu de la vie

Dossier fonctionnel et technique

Introduction

Dans le cadre de notre cursus à Polytech Lyon, on nous a demandé de programmer un jeu de la vie avec différentes options.

Le jeu de la vie, automate cellulaire imaginé par John Horton Conway en 1970, est probablement, à l'heure actuelle, le plus connu de tous les automates cellulaires.

Malgré des règles très simples, le jeu de la vie permet le développement de motifs extrêmement complexes.

Le jeu se déroule sur une grille à deux dimensions, théoriquement infinie (mais de longueur et de largeur finies et plus ou moins grandes dans la pratique), dont les cases — qu'on appelle des « cellules », par analogie avec les cellules vivantes — peuvent prendre deux états distincts : « vivantes » ou « mortes ».

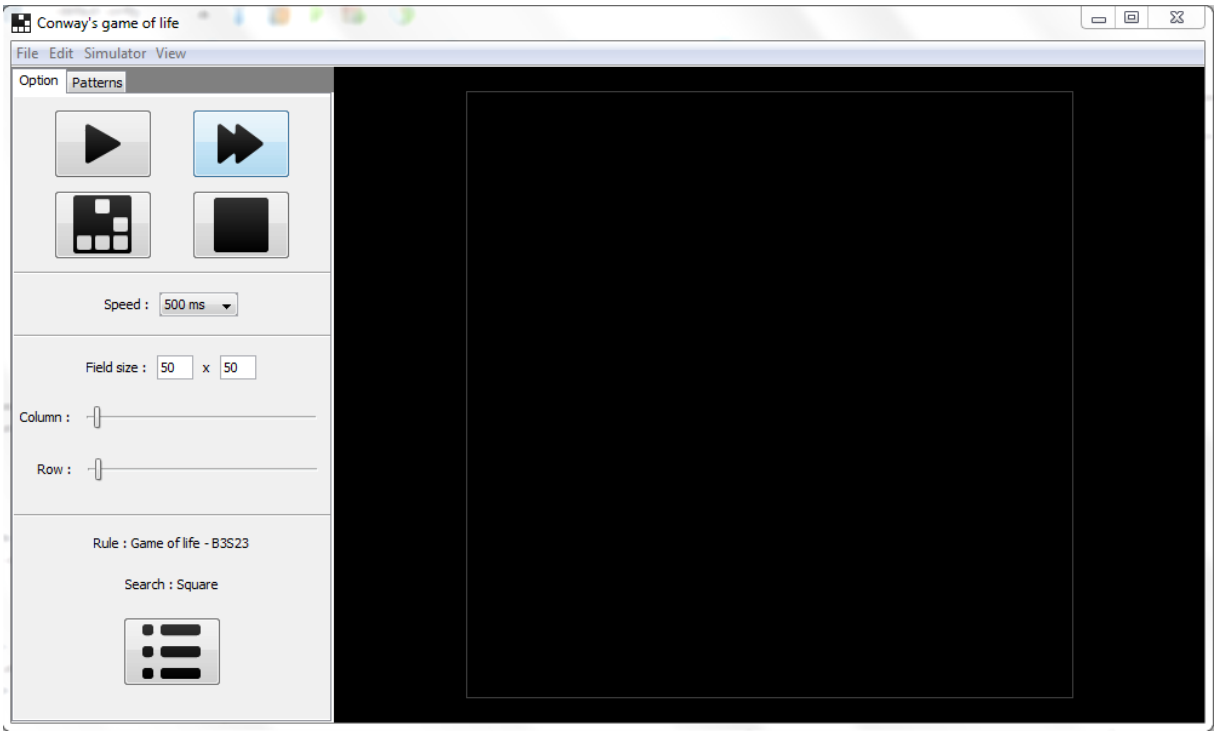
À chaque étape, l'évolution d'une cellule est entièrement déterminée par l'état de ses huit voisines de la façon suivante :

- Une cellule morte possédant exactement trois voisines vivantes devient vivante (elle naît).
- Une cellule vivante possédant deux ou trois voisines vivantes le reste, sinon elle meurt.

Nous avons implémenté un automate cellulaire permettant de jouer au jeu de la vie ainsi que de modifier les règles de déroulement.

Dossier fonctionnel

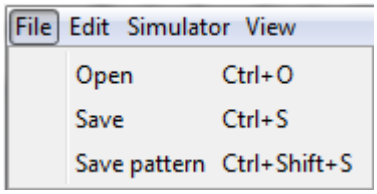
Voici notre interface :



1. La barre de menu

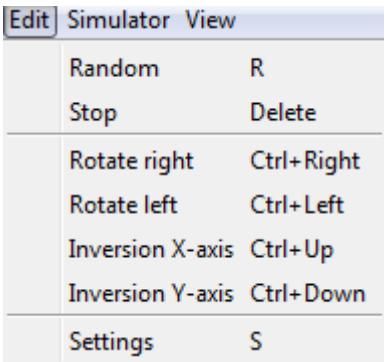
La barre de menu permet d'accéder aux différentes fonctionnalités et de montrer les différents raccourcis. En plus des possibilités offertes par le panel, elle propose aussi l'option de sauvegarde, de chargement et de sauvegarde d'un pattern.

a) Menu File



La fonction « Open » ouvre un sélecteur de fichier qui sera chargé pour l'inclure dans le plateau de jeu.
La fonction « Save » permet de choisir l'emplacement et le nom de fichier à sauvegarder à l'aide d'un sélecteur d'emplacement.
La fonction « Save pattern » demande de saisir le nom du pattern en cours et l'enregistre dans le dossier relatif aux patterns. Elle actualise ensuite la liste des patterns pour l'afficher dans celle-ci.

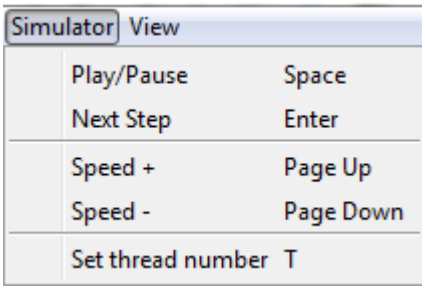
b) Menu Edit



La fonction « Random » permet de remplir la zone de jeu par des cellules vivantes. Ces cellules sont générées aléatoirement.
La fonction « Stop » permet d'arrêter l'exécution du jeu et de vider le plateau.
Les fonctions « Rotate » et « Inversion » permettent de modifier les patterns à ajouter. Nous reviendrons sur ces fonctionnalités dans la partie sur les patterns.

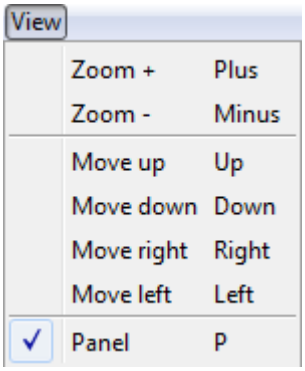
La fonction « Settings » ouvre la fenêtre qui permet de paramétrer les règles du jeu de la vie.

c) Menu Simulator



La fonction « Play/Pause » lance ou arrête l'exécution du jeu.
« Next Step » applique les règles une fois. Cela permet de voir l'étape suivante.
Les menus « Speed » augmentent ou diminuent la vitesse du jeu.
Le menu « Set thread number » permet de modifier le nombre de threads utilisés dans l'application.

d) Menu View



Les fonctionnalités dans cette rubrique permettent de modifier l'affichage. Les fonctions « Zoom » permettent de zoomer sur la zone de jeu. Les fonctions de « Move » déplacent la vue de la zone de jeu dans la direction choisie.
Le menu « Panel » permet d'afficher ou de masquer le panel de l'application (fenêtre qui contient les boutons et paramètres).

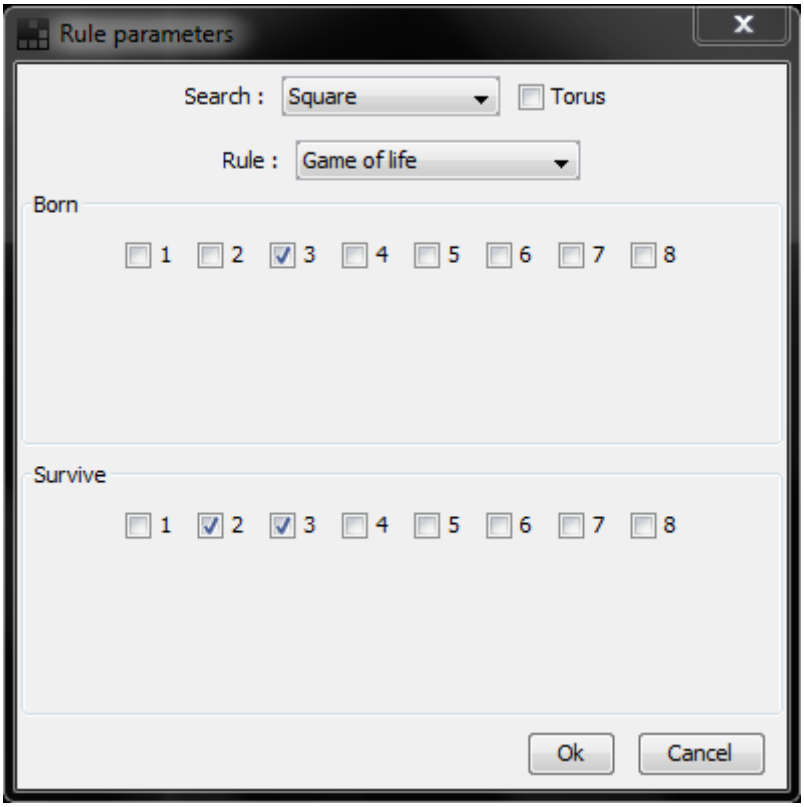
2. Le panel Option

Cette zone de panel permet de faire la gestion du jeu de la vie. Les fonctionnalités sont les mêmes que celles accessibles du menu barre. Pour cette raison nous ne les détaillerons pas dans cette partie.

Cependant on peut constater qu'une partie est dédiée à la taille du plateau de jeu. Elle permet de faire varier le nombre de colonnes et le nombre de lignes à l'aide d'un chiffre ou d'une barre coulissante. Ces valeurs peuvent aller de 1 à 999.

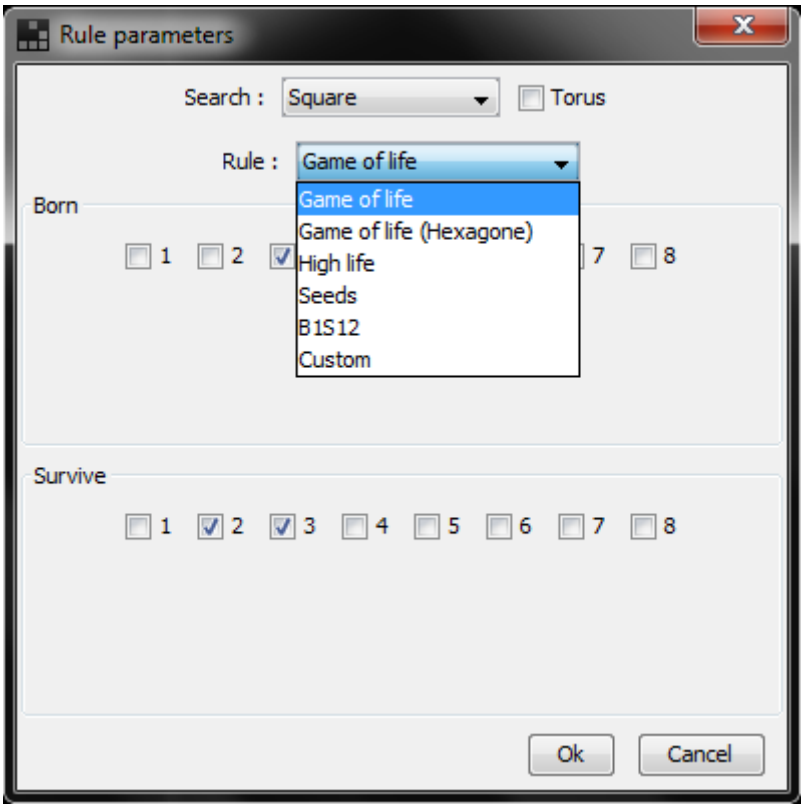
On peut voir aussi qu'une zone au-dessus du bouton « Settings » permet de visualiser les règles du jeu en cours et le type de recherche choisi.

3. La fenêtre « Settings »

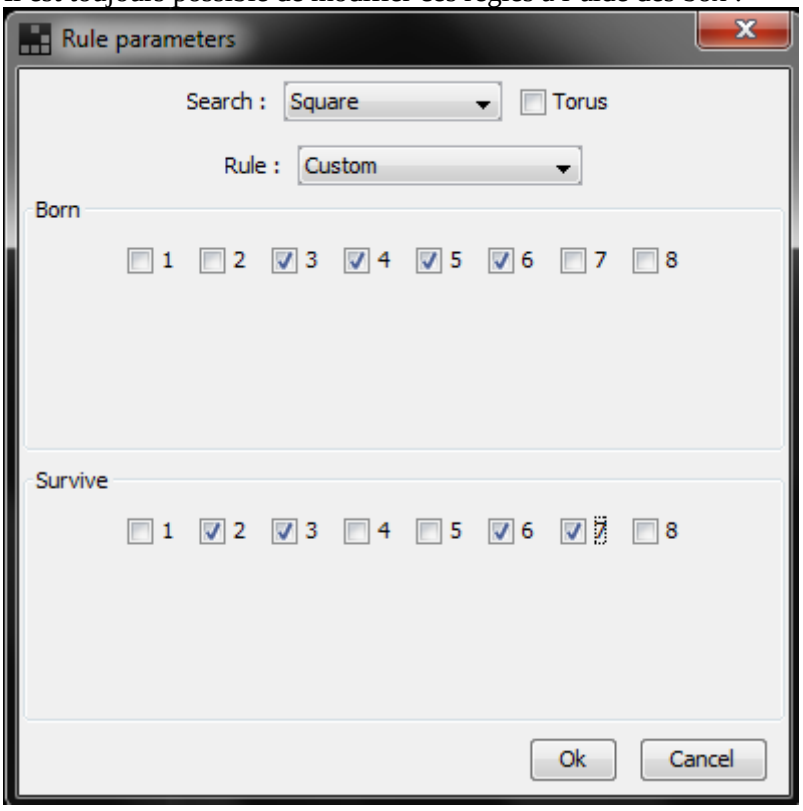


Cette fenêtre est accessible par la barre de menu, le bouton présent dans le panel de l'application ainsi que pas le raccourci S.
Elle permet de choisir les options du jeu de la vie : les règles de survie, de naissance ainsi que le type de recherche.

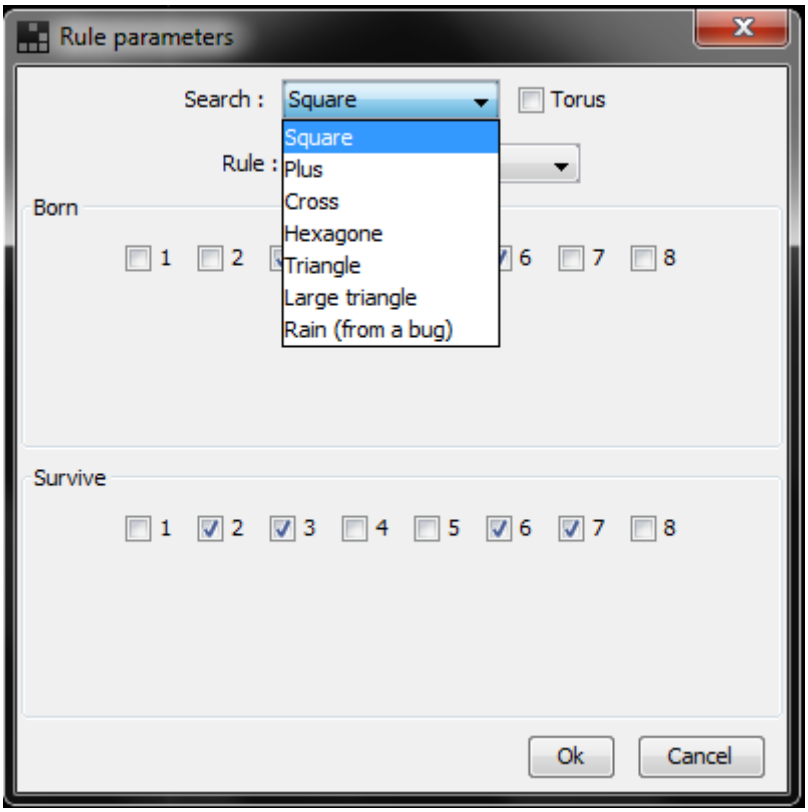
Certaines règles connues sont préenregistrées et permettent de configurer le jeu pour qu'elles soient utilisables.



Il est toujours possible de modifier ces règles à l'aide des box :

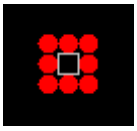


De plus, l'application permet de choisir son type de recherche. Plusieurs ont été implémenté ainsi que plusieurs styles de structures : carré, hexagonale ou encore triangulaire.



a) Square

Cette recherche récupère les voisins qui partagent un sommet (c'est-à-dire les huit cases qui sont autour de la cellule choisie). C'est le type de recherche standard du jeu de la vie.
Voici une image qui illustre cela (La case encadrée correspond à la cellule testée) :

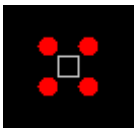


b) Plus

Cette recherche récupère les voisins qui partagent un coté avec la cellule choisie.
Voici l'image qui illustre la recherche « Plus » :

c) Cross

Cette recherche récupère les voisins qui sont directement dans la diagonale de la cellule choisie.
Voici l'image qui illustre la recherche « Cross » :



d) Hexagone

Ce type de recherche modifie l'agencement des cellules pour passer le plateau de jeu en mode hexagone (Voir dans la partie suivante qui présenter les types d'affichages). Elle récupère les six cellules qui sont directement collées à la cellule choisie.
Voici l'image qui illustre la recherche « Hexagone » :

e) Triangle

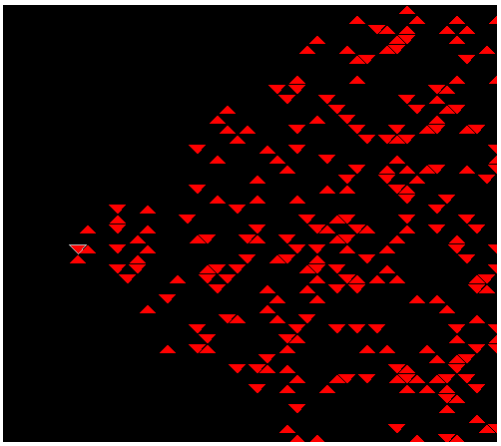
Ce type de recherche modifie l'agencement des cellules pour passer le plateau de jeu en mode triangle (Voir dans la partie suivante qui présenter les types d'affichages). Elle récupère les trois cellules qui partagent un côté avec la cellule choisie.
Voici l'image qui illustre la recherche « Triangle » :

f) Large Triangle

Ce type de recherche modifie l'agencement des cellules pour passer le plateau de jeu en mode triangle (Voir dans la partie suivante qui présenter les types d'affichages). Elle récupère les douze cellules qui partagent un sommet avec la cellule choisie.
Voici l'image qui illustre la recherche « Large Triangle » :

g) Rain

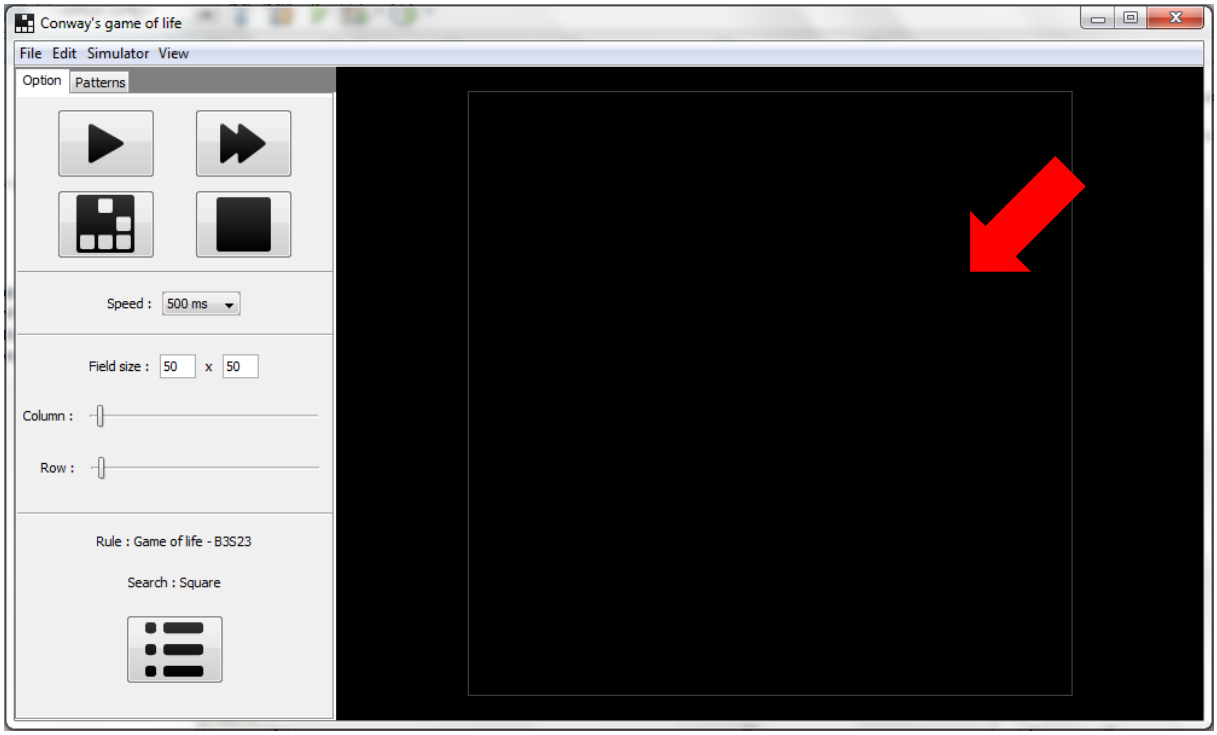
Ce type de recherche est né d'un bug dans notre création de « Large Triangle ». Nous l'avons conservé car quand on maintient le clic appuyé et qu'on monte et descend la souris dans la zone de jeu pendant l'exécution du jeu avec une vitesse suffisamment élevée, cela fait un effet intéressant.
Illustration de cet effet :



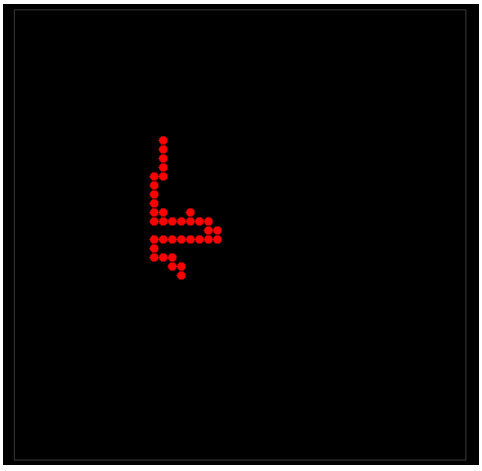
Dans la fenêtre de paramétrage, il est aussi possible de choisir si on souhaite appliquer l'effet de torus au terrain ou non.

Il est également possible de modifier les règles pendant l'exécution du jeu. On peut donc modifier à la volée les conditions de jeu pendant l'exécution de celui-ci.

4. La fenêtre de simulation

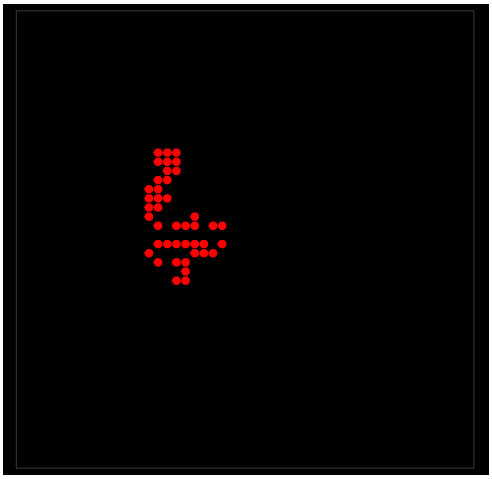


Cette zone permet de visualiser l'évolution du jeu de la vie.
En cliquant à l'intérieur on peut rajouter des cellules ou en retirer (en cliquant sur une cellule qui est déjà vivante).

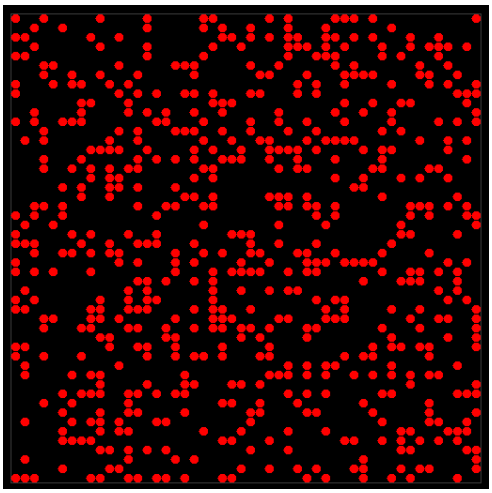


Avec le clic droit (maintenu) ou encore les touches directionnelles, on peut se déplacer dans le plateau pour pouvoir visualiser différents lieux. Il est également possible de zoomer ou dézoomer à l'aide de la molette de la souris, du menu barre ou encore des touches + et -.

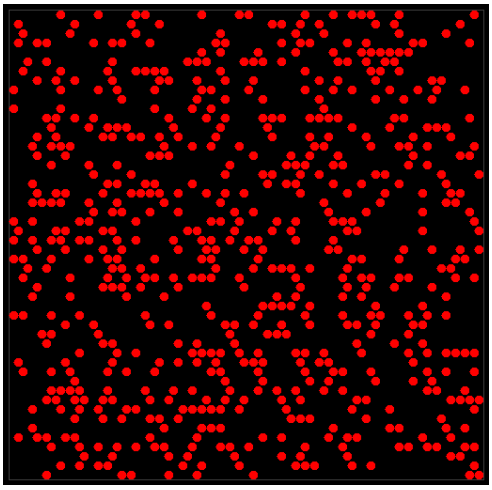
On peut ensuite exécuter le jeu et visualiser les différentes étapes qui s’affichent à l’écran.



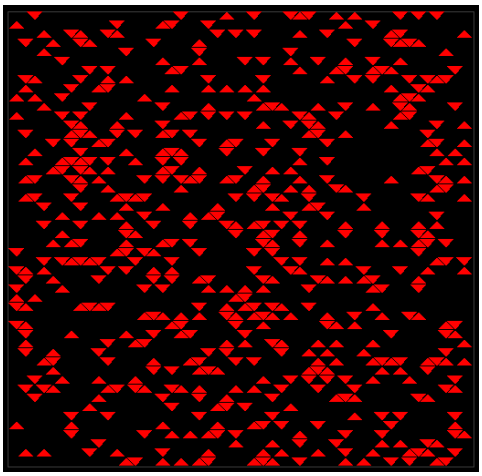
Il est également possible de le remplir à l’aide du bouton random :



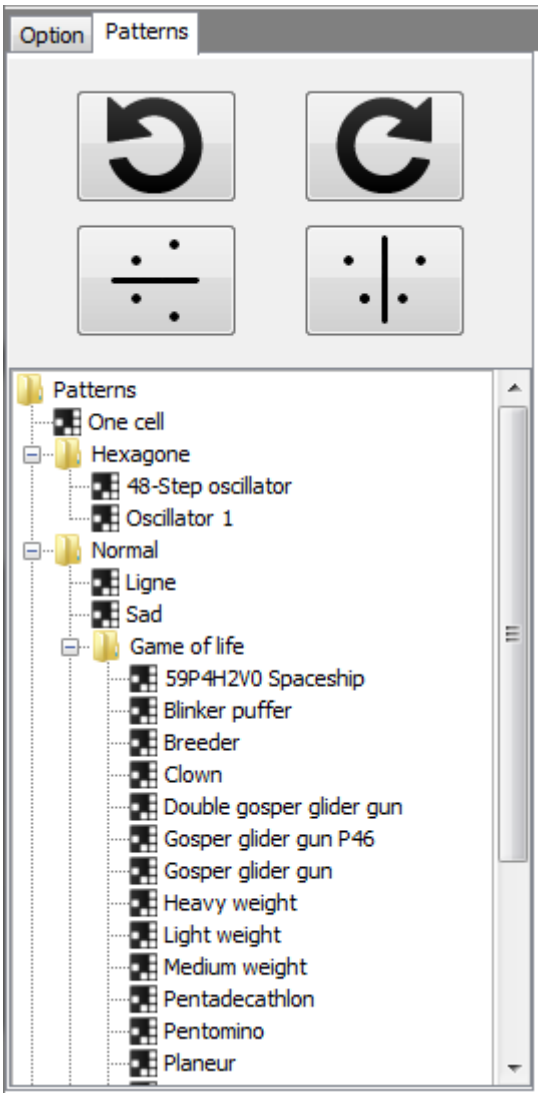
Ce type d’affichage était l’affichage standard sous forme de matrice.
Cependant il est possible dans l’application d’avoir un affichage sous forme hexagonale,



et aussi sous forme triangulaire.



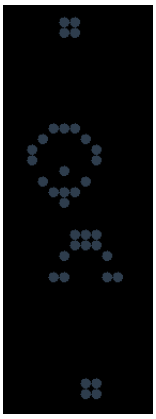
5. Le panel Patterns



Ce panel permet de sélectionner un pattern pour pouvoir l'inclure dans la zone de jeu.
Les différents patterns sont triés par catégorie. En effet, les patterns ne fonctionnent pas dans tous les types de règles.
Quand un pattern est sélectionné, au survol de la zone de jeu il est affiché pour visualiser le positionnement avant l'insertion.
Il est possible d'effectuer des rotations dans le sens des aiguilles d'une montre ou dans le sens inverse à l'aide des boutons ou des raccourcis. Il est également possible de faire une symétrie verticale ou une symétrie horizontale.
Il est possible de désélectionner le pattern en cliquant sur « One cell ».
La liste des patterns est mise à jour après l'enregistrement d'un pattern avec l'option du menu barre.
Cette fonctionnalité prend l'espace de jeu courant et l'enregistre dans les patterns. Après cela, on peut trouver le pattern généré dans le dossier « User » (voir la suite).

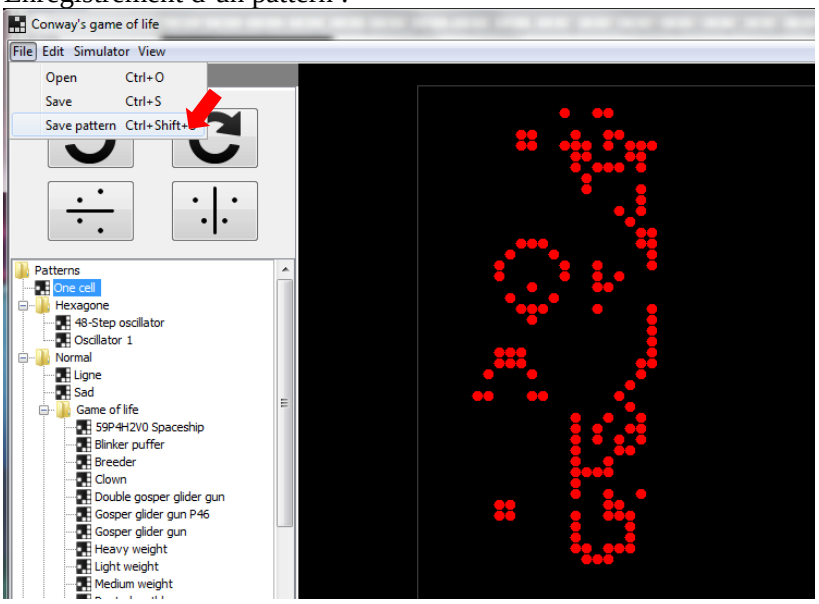
Affichage du pattern au survol :

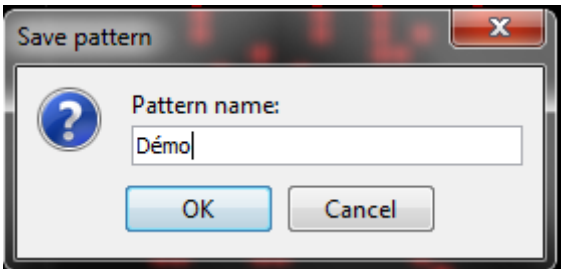
Rotation :

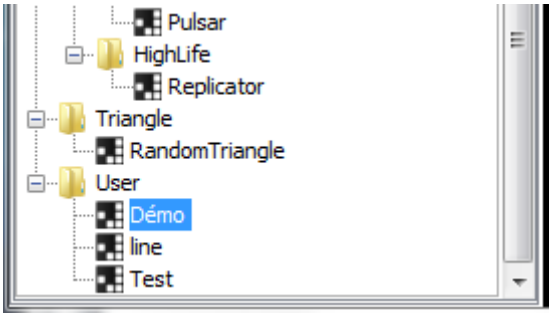


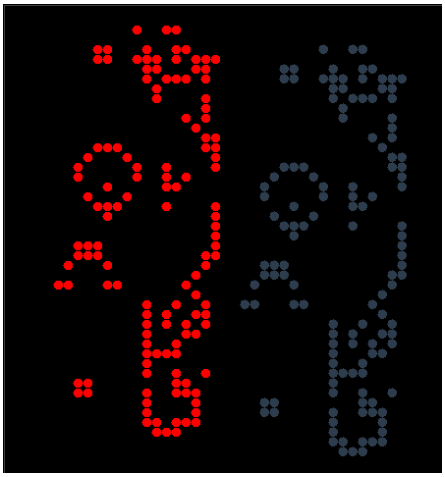
Symétrie :

Enregistrement d’un pattern :

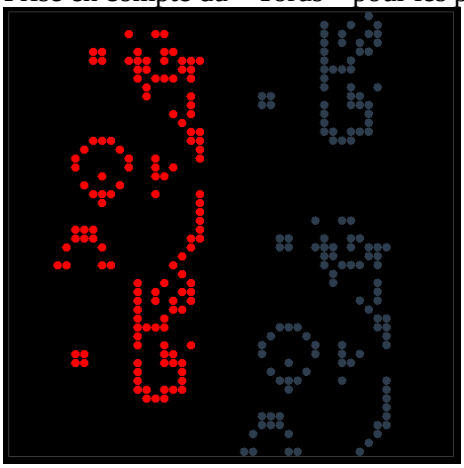








Prise en compte du « Torus » pour les patterns (si l'option est sélectionnée) :



Dossier technique

Notre programme est composé de plusieurs packages qui permettent son bon fonctionnement. Les classes ont été réparties à travers ces packages de façon à créer une application la plus maintenable et évolutive possible.
Nous retrouvons donc trois packages principaux permettant la mise en place du patron de conception MVC :

- model
- view
- controller

Les classes du modèle gèrent toutes les données, le jeu de la vie et l'évolution de celui-ci.
Les classes de la vue mettent en place l'interface graphique pour le dialogue avec l'utilisateur.
Les classes du contrôleur mettent en relation celle de la vue et du modèle el afin de mettre la vue à jour en fonction du modèle et de donner des instructions au model en fonction des actions utilisateur.
Pour plus amples informations concernant le fonctionnement précis de certaines classes, veuillez vous référer à la javadoc de l'application.

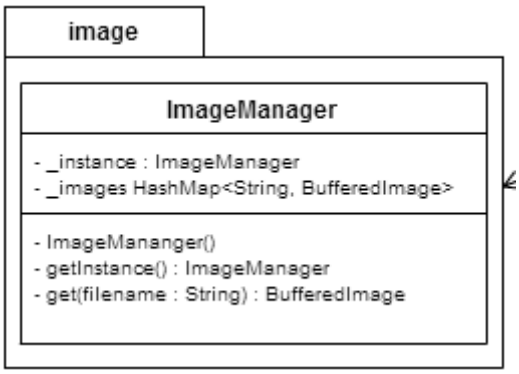
1. Package model

Le modèle gère toutes les données, le jeu de la vie et l'évolution de celui-ci.
Il se décompose en plusieurs sous packages :

- image
- simulator
- gameoflife

Chacun d'eux fournit un module générique pouvant être utilisé indépendamment des autres.

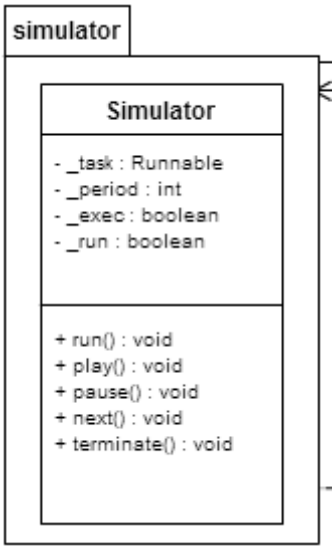
a) Package model.image



Le package model.image fournit une classe ImageManager. Cette classe permet de gérer tous les chargements d'images de l'application.

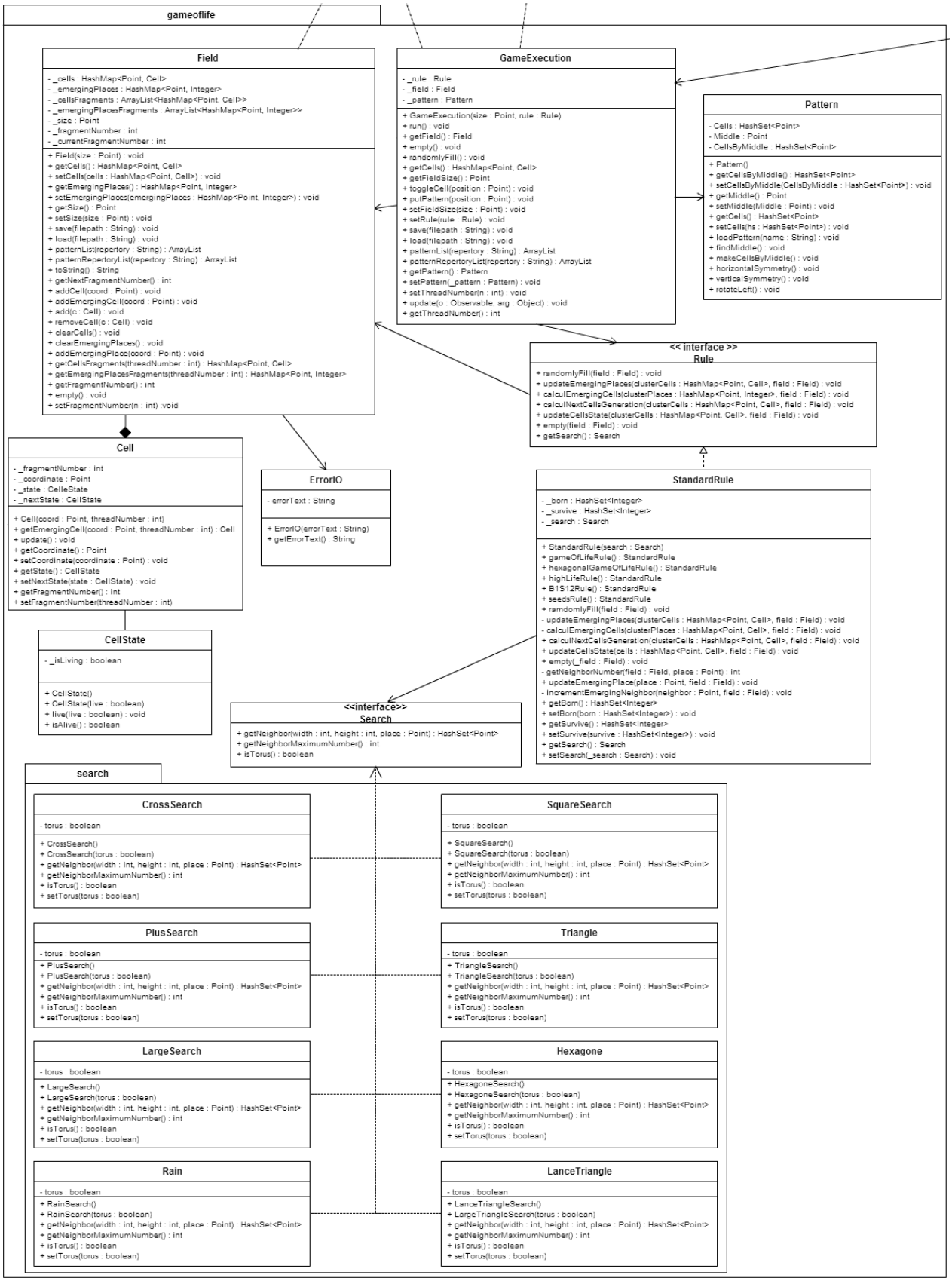
On lui demande de renvoyer une image à partir de son chemin d'accès : si l'image n'a pas été chargé, elle la charge, la stock en mémoire puis la renvoie, sinon elle renvoie juste l'occurrence qu'elle avait gardé en mémoire.
Le but étant de charger qu'une seul fois une image plutôt que de perdre du temps sur le chargement à chaque fois que l'image doit être utilisée.

b) Package model.simulator



Le package **model.simulato**r fournit une classe **Simulator**. Cette classe permet d'effectuer une tâche à une fréquence donnée dans un thread autonome. La classe **Simulator** hérite de la classe **Thread**, cela implique que lorsque le thread se termine on ne peut plus le relancer. C'est pour cela que la méthode **run** est redéfinie pour gérer plusieurs appels à la tâche. Cette tâche doit implémenter l'interface **Runnable** afin de pouvoir être utilisé par le simulateur.
Pour utiliser ce simulateur il faut l'instancier en définissant la tâche à exécuter ainsi que la période. La tâche sera alors exécuté par l'intervalle de temps correspondant à la période. Il est aussi possible de stopper l'exécution (l'exécution de la tache se termine mais ne se relance pas), de la relancer ou simplement de la faire une seule fois. On peut aussi terminer le thread.

c) Package model.gameoflife



Le package **model.gameoflife** contient toutes les classes gérant les données et méta données liées au jeu de la vie.

Les classes **Field** et **Cell** contiennent les métadonnées du jeu de la vie. Autrement dit elle s'occupe de garder en mémoire l'état du terrain et des cellules et aussi l'appartenance des cellules aux threads de calcul.
La classe **Field** garde les cellules en mémoire grâce à une **HashMap**. Cela permet de ne garder en mémoire uniquement les cellules vivantes (c'est une optimisation importante surtout quand on a un grand terrain avec peu de cellules). La classe contient une autre **HashMap** permettant de stocker les endroits voisins des cellules vivantes car une cellule peut potentiellement naître sur ces cases. Enfin la classe contient une **ArrayList** de **HashMap** afin de répartir les cellules dans autant de **HashMap** que de thread de calcul. Ainsi le calcul de la génération suivante sera partitionné et chaque partition ne travaillera que sur une partie des cellules. Le même principe est appliqué pour les endroits voisins des cellules vivantes. C'est donc cette classe qui indique le nombre de thread de calcul.
Cette classe contient aussi les fonctions pour enregistrer et charger un terrain.
La classe **Cell** permet de garder en mémoire l'état d'une cellule ainsi que son numéro de thread associé.

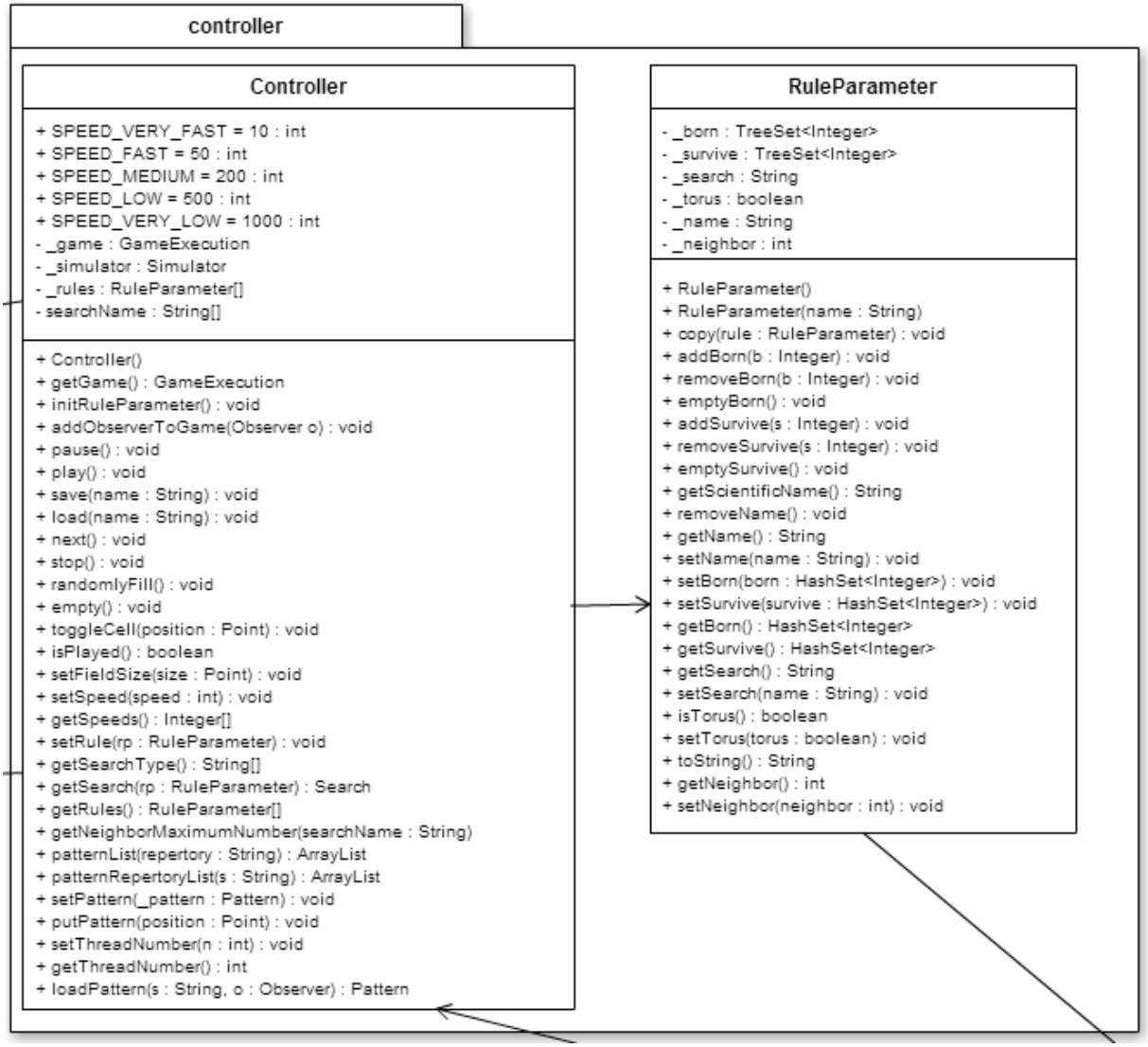
Les classes **Rule**, **Search** et **CellState** définissent le comportement du jeu. La classe **CellState** gère l'état d'une cellule. Elle contient sa position et permet de savoir si la cellule est en vie ou non. En héritant de cette classe on peut facilement ajouter des états aux cellules comme de la vie.
L'interface **Rule** fournit une interface dont le but est de gérer les générations de cellules : vider le terrain, le remplir aléatoirement ou passer d'une génération à une autre. En l'implementant on définit donc les règles du jeu.
Les fonctions pour passer d'une génération à une autre prennent deux paramètres : le groupe de cellules sur lequel travailler ainsi que le terrain.
Le passage d'une génération à une autre se répartit en plusieurs fonction. Tout d'abord on met à jour les voisins des cellules vivantes (fonctions **updateEmergingPlaces** et **calculEmergingCells**) ensuite on calcule pour chaque cellule si elle doit mourir ou survivre (**calculNextCellsGeneration**) et enfin on met à jour l'état de toutes les cellules.
La classe **StandardRule** implémente la classe **Rule** pour fournir les règles standards des automates cellulaires : on peut changer le nombre de voisines dont une cellule a besoin pour survivre ou naître.
Enfin la classe **Search** est une interface dont le but est de fournir des méthodes permettant de récupérer les cases voisines d'une cellule donnée. Cela permet de pouvoir implémenter plusieurs sortes de terrain. Par exemple, si une cellule est sur le bord du terrain et que la classe récupère les cases situé de l'autre coté du terrain alors on obtient un tore. On peut aussi générer un terrain à cases hexagonales (en ne prenant que 6 voisins).
Le package **model.gameoflife.search** contient différentes classes implémentant l'interface **Search**. On ne reviendra pas dessus ici car leurs fonctionnalités ont déjà été expliquées précédemment.

Dans ce package on trouve aussi la classe **GameExecution**. Cette classe permet de centraliser le dialogue entre le controleur et le modèle afin que le controleur ne puisse pas faire n'importe quoi dans le modèle. Cette classe implémente l'interface Runnable, elle permet donc d'être utilisé par le simulateur pour passer d'une génération de cellules à une autre. Elle gère ainsi la répartition des threads de calculs grâce à ses classes internes.

Ensuite, la classe **Pattern** gère le modèle de cellules que l'on peut ajouter sur le terrain. Notamment la sauvegarde et le chargement de ces modèles.

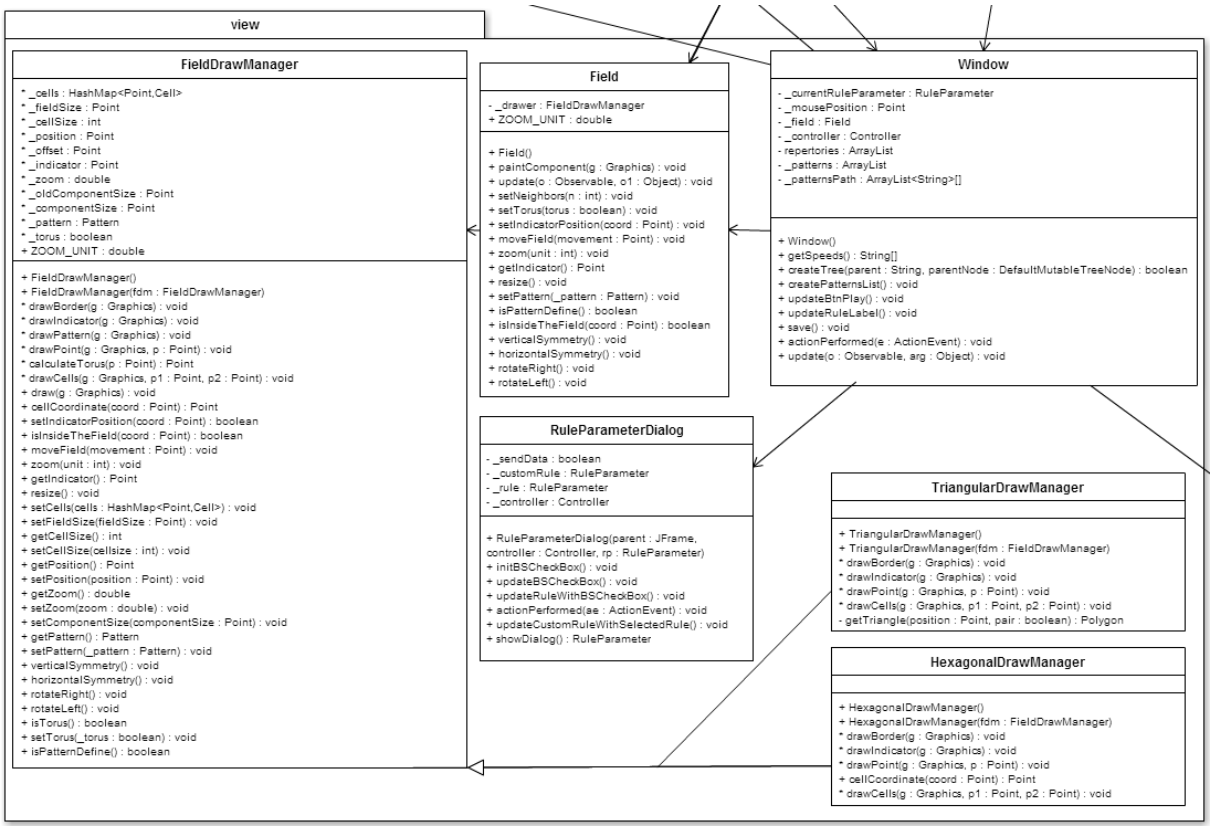
Enfin la classe **ErrorIO** est une exception dont le but est de faire remonter à l'utilisateur les erreurs liées aux flux fichier.

2. Package controller



Ce package contient deux classes : **Controller** et **RuleParameter**. Le rôle de cette dernière est de contenir les règles que l'utilisateur a spécifiées pour que le contrôleur puisse donner les bonnes indications au modèle.
La classe controller quant à elle s'occupe de donner les ordres aux modèles en fonction des entrées utilisateurs. Elle s'occupe aussi d'instancier le modèle et de gérer le simulateur.

3. Package view.



Le package view contient les classes servant à l'interface homme/machine de l'application.

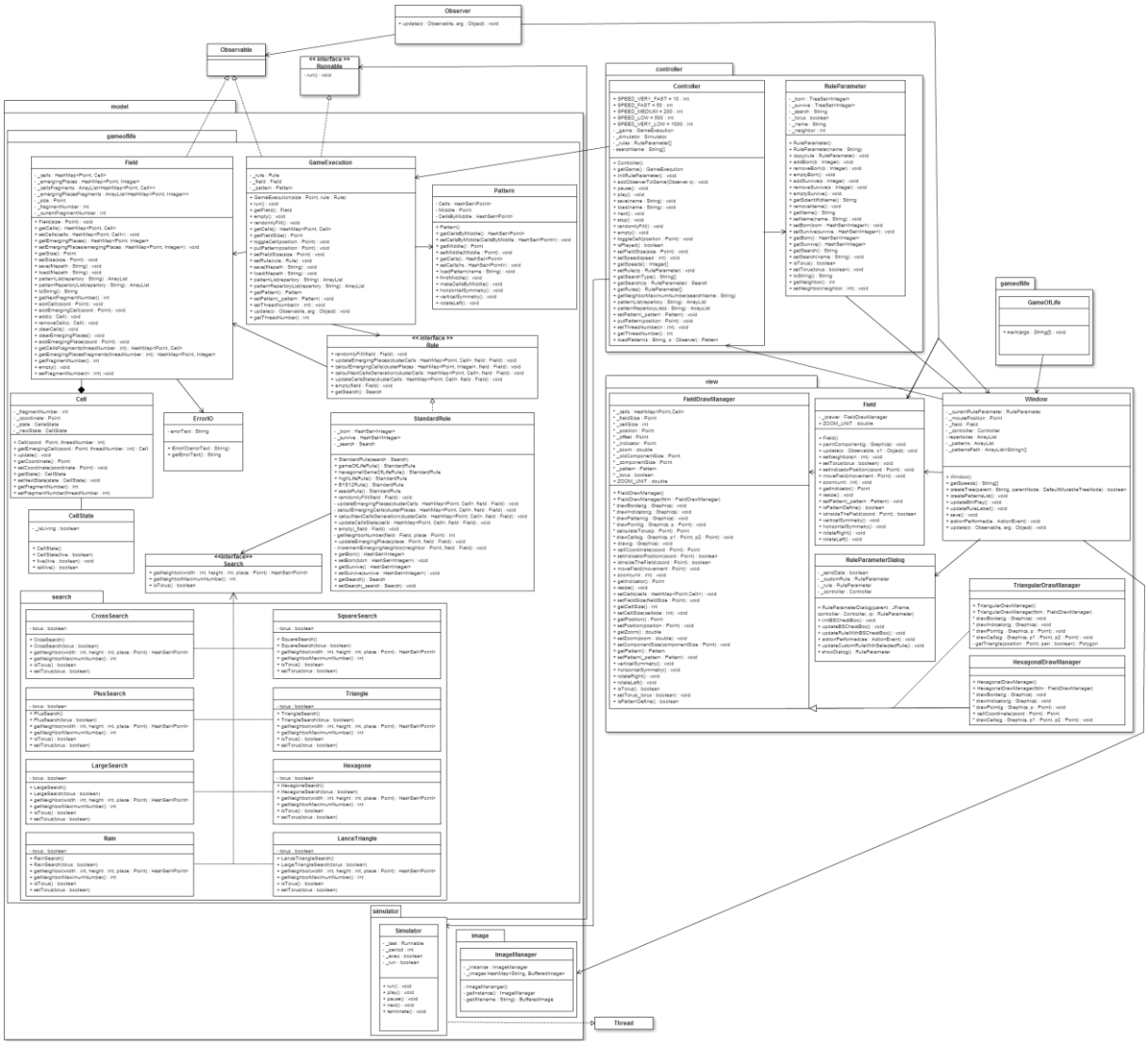
On ne reviendra pas sur les fonctionnalités car elles ont déjà été expliquées précédemment.

La classe **Window** gère l'affichage de la fenêtre principale de l'application. Elle contient notamment le composant graphique **Field**. Celui-ci s'occupe de l'affichage du terrain. Cette classe observe la classe **model.gameoflife.Field**, ainsi lorsque le terrain change, l'affichage est changé en même temps.

Pour l'affichage du terrain, la classe **Field** fait appel à la classe **FieldDrawManager** pour dessiner le terrain. En effet, exporter la gestion du dessin du terrain dans une classe, qui n'est pas un composant graphique, permet de changer cette gestion très simplement en créant une classe héritant de celle-ci. Ainsi on peut prévoir plusieurs affichages différents et les changer à la volée juste en changeant l'objet utilisé par la classe **Field**.

Ainsi les classes **HexagonalDrawManager** et **TriangularDrawManager** fournissent le dessin de terrain à cellules hexagonales ou triangulaires.

Annexe : Diagramme de classe



(L’image est disponible en taille réelle)