Université Abou Bekr Belkaied

Faculté de science

Département informatique

Jeu de la vie

Conception et réalisation

En 1970 , John Conway a inventé ce jeu, le jeu est basé sur un approche mathématique et les mathématiciens cherchent encore ses secrets. Dans cet article nous faisons une petite démonstration sur le jeu, sa conception et sa réalisation sous une application Java.

Mohammed Nadir BELARROUCI
13/12/2015

Jeu de la vie ?

Quand on dit « jeu » signifie qu'il ya au moins un joueur et pour cela on va préciser que le Jeu de la vie n'est pas vraiment un jeu au sens ludique, puisqu'il ne nécessite aucun joueur.

Donc c'est quoi au juste ce fameux jeu ?

C'est une automate cellulaire, disant un univers parfaitement déterministe où il n y a pas du hasard et tout est régler par des règles simples et qui se réplique.

Les automates cellulaires sont les plus petites machines qu'on peut imaginer , un **automate cellulaire** consiste en une grille régulière de « cellules » contenant chacune un « état » choisi parmi un ensemble fini et qui peut évoluer au cours du temps

Ce qui est magique dans ce jeu c'est qu'il a la fois simple et extraordinaire

Le jeu déroule sur une grille de deux dimensions théoriquement infinie dont les cases qu'on appelle des cellules et qui peuvent être dans deux états distincts « vivants » ou «mortes ».

Pour passer d'une configuration a une autre on utilise les règles suivantes :

- 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.

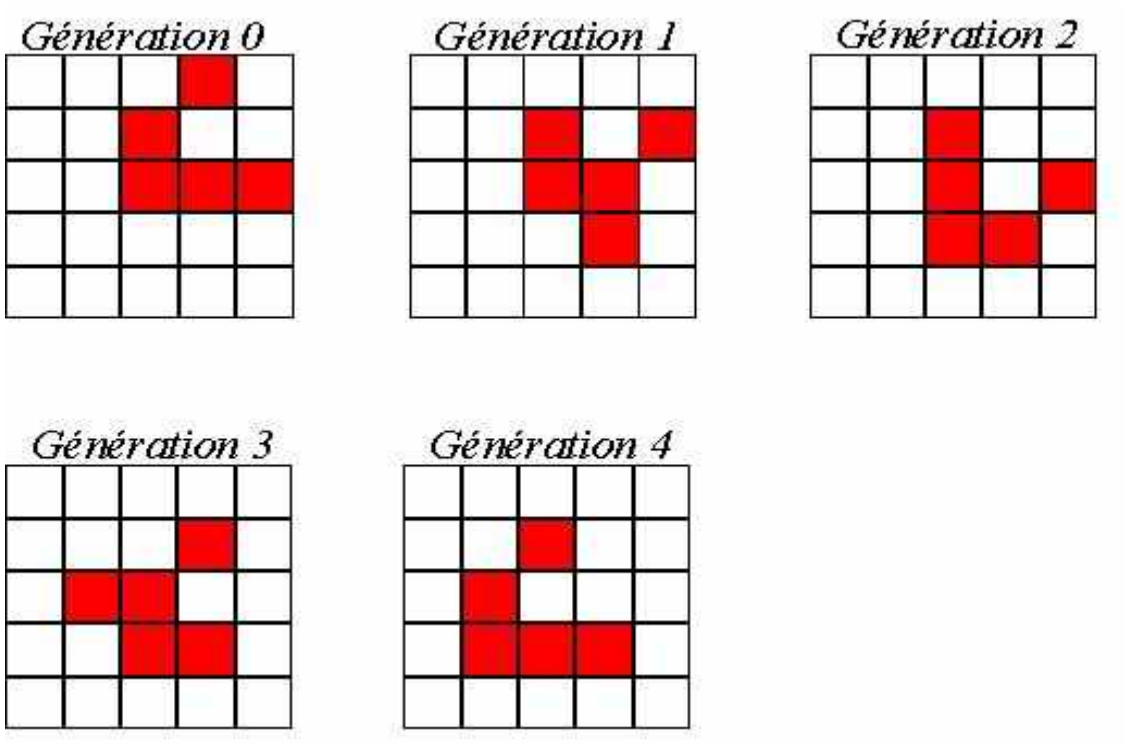

Figure1 : 5 génération de l'évolution d'un planeur .

Les mathématiciens recherchent encore et encore pour des configurations disant un peu spécial , comme le planeur , les vaisseaux , le canon, le jardin d'Aden , certaines entre eux consiste de calculer les nombres premiers , dernièrement il ont crée une configuration pour calculer les nombres décimaux du nombre PI.

Dés maintenant il reste toujours de troues noires et des questions qui ne sont pas encore démontrées.

Passant maintenant aux choses sérieuses la conception et la réalisation du jeu.

Algorithme

Il y a plusieurs algorithmes pour résoudre ce problème, d'ailleurs la simplicité de ses règles va nous facilite les taches, avant commencer je veux mentionner qu'il existe déjà des programmes très puissant (Golly par exemple) construit dans le but d'étudier ce jeu et ils utilisent des algorithmes spécial (rapide aux termes de complexité) puisque ils travaillent sur des matrices très grand.

Et voilà, nous y sommes

```
Fonction génarationSuivante(a), a : matrice
Var :
        nbrCellule : entier
        ListCellule : List de cellule , cellule = {case c , état e}
Début
        Pour chaque case « c » dans « a »  faire
                nbrCellule = calculerNbrVoision() ;
                si nbrCellule < 2 OU nbrCellule > 4 ET « c »vivante
                        listCellule.ajouter(c,morte) ;
                sinon si nbrCellule == 3 ET c est morte
                        listCellule.ajouter(c,naissance) ;
                fin sinon si
        fin pour
        pour chaque Cellule x dans listCellule
                si  x.e = naissance
                        a(x.c) = vivante ;
                sinon
                        a(x.c) = mourante ;
fin.
```

Au premier temps j'ai penser d'utiliser deux matrices une pour la génération courante et l'autre pour la génération suivantes , mais c'était une très mauvaise idée , il va parcourir deux matrice a la places d'une seule , et puisque les matrices sont très grand donc sa va poser beaucoup de problèmes au termes du temps d'exécution, la deuxième idée que on peut penser , est de calculer le nombre de voisins et faire des modifications sur la cellule où on travaille , lorsque on termine toute la matrice , on refait un deuxième parcoure pour avoir la génération suivante , mais on tomber sur le même problème que la 1^{ere} idée , puis on arriver a cet algorithme dont l'idée est de sauvegarder les cellules qui vont changer avec leur état suivante (mort/naissance) et après la fin du parcoure sur la matrice , on cibles ces cellules directement et on met nos modifications nécessaire

En résumé : il y en a surement des algorithmes beaucoup mieux que ça , mais au moins on rassurer que si il n'est pas le meilleur , en parallèle il n'est pas le mauvais .

Dans l'étape suivante on va faire une petite introduction sur l'api SWT (standard widget toolkit) dont notre application java est basée sur. Et

D'autres notion sur la modalisation pour mieux structurer son application

Conception

Donc c'est quoi au juste ce fameuse api SWT ?

La première API pour développer des interfaces graphiques portables d'un système à un autre en Java est AWT. Cette API repose sur les composants graphiques du système sous-jacent ce qui lui assure de bonnes performances. Malheureusement, ces composants sont limités dans leur fonctionnalité car ils représentent le plus petit dénominateur commun des différents systèmes concernés.

Pour pallier ce problème, Sun a proposé une nouvelle API, Swing. Cette Api est presque exclusivement écrite en Java, ce qui assure sa portabilité. Swing possède aussi d'autres points forts, telles que des fonctionnalités avancées, la possibilité d'étendre les composants, une adaptation du rendu de composants, etc ... Swing est une API mature, éprouvée et parfaitement connue. Malheureusement, ses deux gros défauts sont sa consommation en ressource machine et la lenteur d'exécution des applications qui l'utilisent.

Pour cela SWT propose une approche intermédiaire : utiliser autant que possible les composants du système et implémenter les autres composants en Java. SWT est écrit en Java et utilise la technologie JNI pour appeler les composants natifs. SWT utilise autant que possible les composants natifs du système lorsqu'ils existent, sinon ils sont réécrits en pur Java. Les données de chaque composant sont aussi stockées autant que possible dans le composant natif, limitant ainsi les données stockées dans les objets Java correspondant.

Si vous intéressez à apprendre développez ces applications SWT je vous propose de consulte ces deux sites avec pleins de documentations et d'exemples :

- www.**jmdoudoux**.fr/java/dej/chap-**swt**.htm
- www.**java2s**.com/...**SWT**/Catalog0280__**SWT**.htm

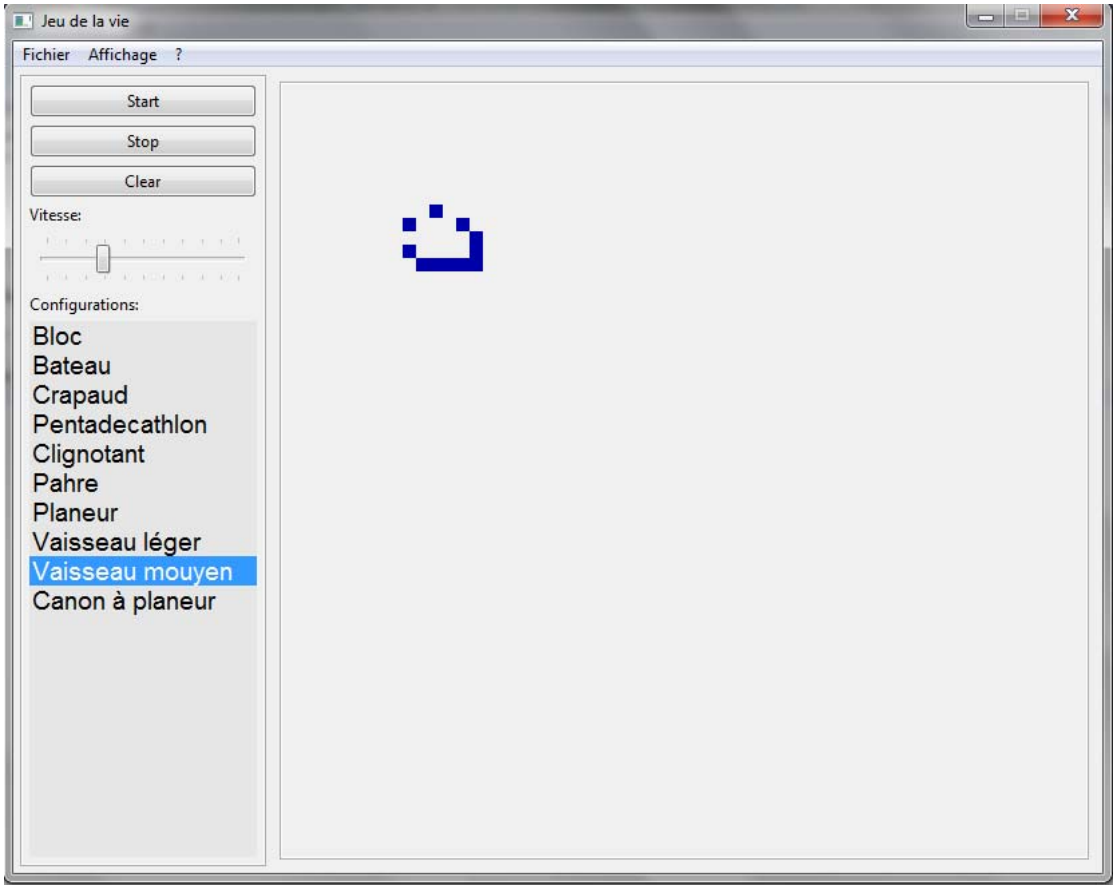

Figure 2 : Jeu de la vie sous une application SWT.

Passant maintenant au model MVC (Model Vue Control) , sa définition, et à quoi sert au juste :

le pattern MVC permet de bien organiser son code source. Il va vous aider à savoir quels fichiers créer, mais surtout à définir leur rôle. Le but de MVC est justement de séparer la logique du code en trois parties que l'on retrouve dans des fichiers distincts, comme l'explique la description qui suit.

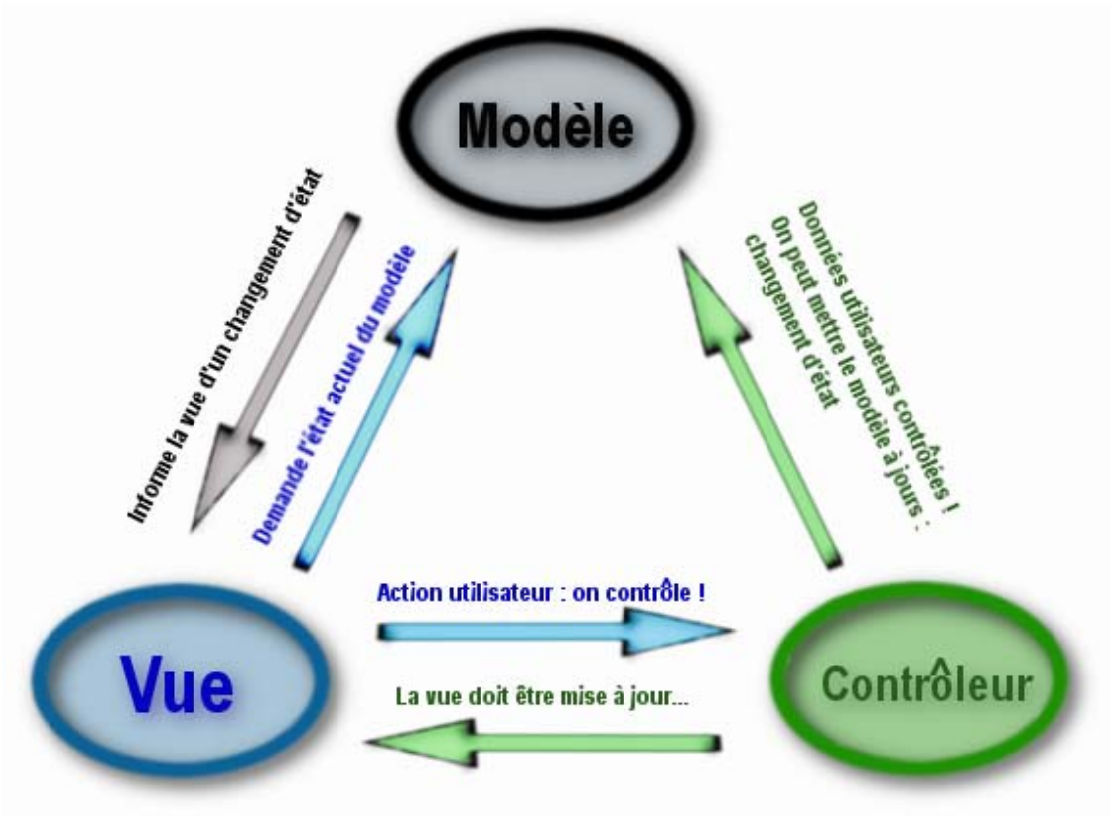

Figure 3 : Model MVC

La vue :

Ce que l'on nomme « la vue » est en fait une IHM. Elle représente ce que l'utilisateur a sous les yeux. La vue peut donc être :

- une application graphique Swing, AWT, SWT pour Java (Form pour C#…) ;
- une page web ;
- un terminal Linux ou une console Windows ;

Le modèle :

Le modèle peut être divers et varié. C'est là que se trouvent les données. Il s'agit en général d'un ou plusieurs objets Java. Ces objets s'apparentent généralement à ce qu'on appelle souvent « la couche métier » de l'application et effectuent des traitements absolument transparents pour l'utilisateur.

Le contrôleur :

Cet objet - car il s'agit aussi d'un objet - permet de faire le lien entre la vue et le modèle lorsqu'une action utilisateur est intervenue sur la vue. C'est cet objet qui aura pour rôle de contrôler les données.

Pour implémenter le MVC en java il vaut mieux aussi avoir une idée sur Pattern Observer/Observable

Ce utilisé pour envoyer un signal à des modules qui jouent le rôle d'observateur. En cas de notification, les observateurs effectuent alors l'action adéquate en fonction des informations qui parviennent depuis les modules qu'ils observent (les « observables »).

La figure suivante est la représentation UML de l'application entière

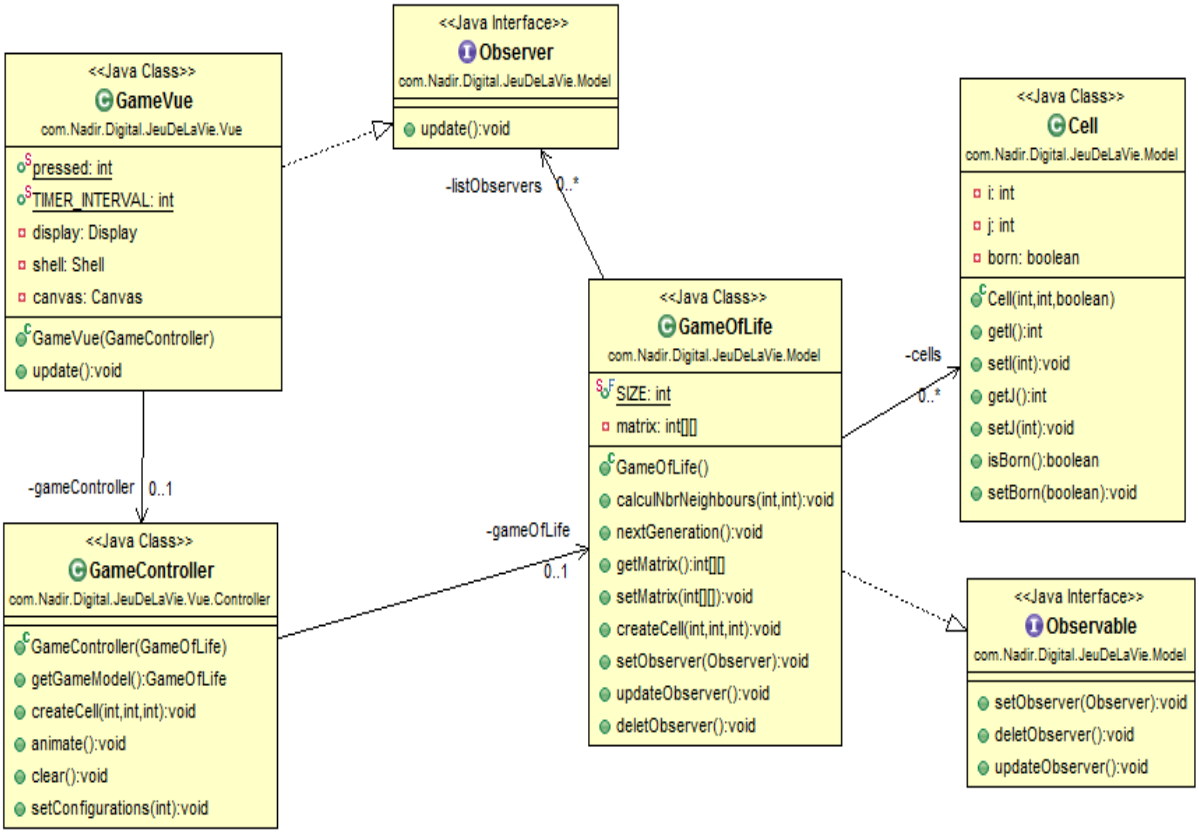

Figure 4 : représentation UML des Classes Java

Model

La classe GameOfLife joue le rôle du model, elle implémente l'interface Observable ,elle passe d'une génération à l'autre grâce à sa méthode nextGeneration() qui utilise notre algorithme précédent, après chaque passage de génération elle appelle la méthode updateObserver() qui va notifier notre Vue (GameVue) .

Code source de la classe :

```
public class GameOfLife implements Observable{

	//la taille du matrice
	public static final int SIZE=60;
	private int matrix[][] = new int [SIZE][SIZE];

	//la list des observateurs qui observe notre model
	private ArrayList<Observer> listObservers = new
ArrayList<Observer>();
	// une liste de cellule celle qui est utilsé dans notre algortithme

	private ArrayList<Cell> cells ;

	/**
	 * la classe à un seul constructeur
	 * son role est d'initialiser la matrice avec une petit droite
	 * NB: une cellule vivante prend la valeur 1 sinon 0
	 * */
	public GameOfLife() {
		cells = new ArrayList<Cell>();

		for(int i=10;i<=12;i++){
			matrix[3][i] = 1;
		}
	}
	/**
	 * cette méthod calcule le nombre de voisin d'une cellule ,
	 * elle applique les règles des jeux , en ajoutant cette cellule a
notre liste soit avec une état mourante ou naissance
	 * @param i position de la cellule dans une colonne
	 * @param j position de la cellule dans une ligne
	 */
	public void  calculNbrNeighbours(int i ,int j){
		int cellNbr = 0;
		try{

	// deux boucle pour calculer le nbr de cellule vivantes dans le
	voisinage
		for(int k=(i==0)?0:-1;(i==(SIZE-1))?k<1:k<=1;k++){
			for(int l=(j==0)?0:-1;(j==(SIZE-1))?l<1:l<=1;l++){

					if(matrix[i+k][j+l]==1){
						if(l!=0 || k!=0){
							cellNbr++;
```

```
                                        }
                                }
                        }
                }


                //application des règles naissance si 3 , survivre si 2
ou 3

                if(cellNbr<2||cellNbr>3 && matrix[i][j] == 1){
                        cells.add(new Cell(i, j, false));
                }else if(cellNbr == 3 && matrix[i][j] == 0){
                        cells.add(new Cell(i, j, true));
                }

        }catch(ArrayIndexOutOfBoundsException e){
                e.printStackTrace();
        }
    }


    /**
     * la méthode nextGeneration applique  l'algorithme
     *
     * */
    public void nextGeneration(){
        // initialisation a liste
        cells = new ArrayList<Cell>();

        // pour chaque on calcule le nbr de voisins
        for(int i=0;i<SIZE;i++){
                for(int j=0;j<SIZE;j++){

                        calculNbrNeighbours(i, j);

                }

        }

        // pour chaque cellule modifié on fait les modification
nécéssaire
        for (Cell cell : cells) {
                if(cell.isBorn()){
                        matrix[cell.getI()][cell.getJ()] = 1;
                }else{
                        matrix[cell.getI()][cell.getJ()] = 0;
                }
        }
        updateObserver();

    }


    /**
     * getter
     * @return matrix
     * */
    public int[][] getMatrix() {
        return matrix;
    }
    /**
     * setter
```

```
	 * @param matrix : matrice
	 * */

	public void setMatrix(int[][] matrix) {
		this.matrix = matrix;
	}

	/**
	 * la méthode creatCell cree ou tu une cellule
	 *
	 * @param i position de la cellule dans une colonne
	 * @param j position de la cellule dans une ligne
	 * @param type
	 * */
	public void createCell(int i,int j,int type){
		try {
			if(type == 1)
				matrix[i][j] = 1;
			else if(type == 3)
				matrix[i][j] = 0;
		} catch (IndexOutOfBoundsException e) {

		}
		updateObserver();
	}


	/**
	 * ajout d'un observateur a notre liste d'observateur
	 * @param obs : Observateur
	 * */
	@Override
	public void setObserver(Observer obs) {
		this.listObservers.add(obs);

	}

	/**
	 * notification de chaque observateur pour qu'il se change
	 * */
	@Override
	public void updateObserver() {
		for (Observer obs : listObservers) {
				obs.update();
		}

	}
	/**
	 * suppression de tout les observateur
	 * */
	@Override
	public void deletObserver() {
		listObservers = new ArrayList<Observer>();

	}
}
```

Vue

La classe GameVue joue le rôle du model, elle implémente l'interface Observer, elle utilise des objets de l'api SWT, elle a aussi un contrôleur dans le but de fournir les informations nécessaire aux model.

Pour l'animation on utilise un thread qui va appeler la méthode animate() de l'objet gameController , ce dernier va appeler la méthode nextGeneration() du model qui va aussi notifier GameVue pour se redessiner .

Le code source de la classe :

```
public class GameVue implements Observer {

	/**
	 * La classe GameVue joue le role de Vue dans notre MVC ,
	 * elle va alterer communiquer avec notre controlleur qui va alterer le model de se changer,
	 * ce dernier va notifier notre vue si il ya des changements
	 *
	 * */
	// la varibale static "pressed" sert a contenir le button pressé par la souris , gauche/droite sinon sa valeur est 0
	public static int pressed = 0;

	// TIMER_INTERVAL define la temps entre chaque appelle du thread de l'animation
	public  static  int TIMER_INTERVAL = 500;


	//l'objet Display pour gerer la pile des evenements
	private Display display;
	// fenetre
	private Shell shell;
	// le canvas qui va etre la tbleau sur qu'on va dessiner sur
	private Canvas canvas;

	// controleur du jeu
	private GameController gameController;

	/**
	 * la creation de l'interface graphique ,
	 *
	 * @param gameController
	 * */
	public GameVue(GameController gameController) {

		// initialisation du controlleur
		this.gameController = gameController;
		this.gameController.getGameModel().setObserver(this);
```

```
            display= new Display();
            shell = new Shell(display,SWT.CLOSE);
            shell.setSize(820,650);
            shell.setLayout(new GridLayout(2,false));
            shell.setText("Jeu de la vie");

            // l'initialisation du thread pour l'animation
            PaintThread  runnable = new PaintThread();

            VueHelper.color = new Color(display, new RGB(0, 0, 168));
            // creation de la barre menu
            Menu menu  = VueHelper.createMenu(shell,canvas,this);


            Composite options = new Composite(shell, SWT.BORDER);
            options.setLayoutData(new GridData(SWT.BEGINNING, SWT.FILL,
false, true, 1, 1));
            options.setLayout(new GridLayout());

            //creation des composant de Panel options
            Button start = VueHelper.createThreadButton(options,
"Start",TIMER_INTERVAL,runnable);
            Button stop = VueHelper.createThreadButton(options,"Stop",-
1,runnable);
            Label label = VueHelper.createLabel(options,"Vitesse: ");
            Scale scale = VueHelper.createScale(options);
            label = VueHelper.createLabel(options, "Configurations: ");
            List list = VueHelper.creatLsitConfigurations(options);



            Composite canvasComposite = new Composite(shell, SWT.NONE);
            canvasComposite.setLayoutData(new GridData(SWT.FILL, SWT.FILL,
true, true, 1, 1));
            canvasComposite.setLayout(new GridLayout());

            // creation de canvas
            canvas = VueHelper.createCanvas(canvasComposite, new Paint(),
new ClickEditMode(), new MoveEditMode());

            // ouveture de la fenetre
            shell.open();
            // la boucle des evenements
            while(!shell.isDisposed()){
                  if(!display.readAndDispatch()){
                        display.sleep();
                  }
            }

            // libérations de ressources
            display.timerExec(-1, runnable);
            display.dispose();

      }

      /**
       * une class interne qui est implimante l'interface Runnable
       * dans le but de appeler la méthode animate celle de l'objet
gameController
       *
```

```
         * */
        class PaintThread implements Runnable{


                public void run() {
                      gameController.animate();
                        display.timerExec(TIMER_INTERVAL, this);
                      }

        }


        /**
         * une class interne qui est implimante l'interface PaintListener
         * dans le but de redessiner si des changement sont effectuer par le
model
         *
         * */
        class Paint implements PaintListener{

              @Override
              public void paintControl(PaintEvent e) {
                    // onrecupere la matrice de Model
                    int matrix[][] =
gameController.getGameModel().getMatrix();


                    for(int i=0;i<GameOfLife.SIZE;i++){
                          for(int j=0;j<GameOfLife.SIZE;j++){
                                // mettre la couleur de fond en blue
                                e.gc.setBackground(VueHelper.color);

                                // si il ya une cellule vivante on affiche
un carré bleu
                                if(matrix[i][j] == 1)
                                      e.gc.fillRectangle(j*10, i*10, 10,
10);
                          }
                    }
              }

        }

        /**
         * une class interne qui est implimante l'interface MouseListener
         * dans le but de crrer un cellule ou bien la supprimer
         * elle jour le role d'un mode editeur
         * */
        class ClickEditMode implements MouseListener{
              @Override
              public void mouseDoubleClick(MouseEvent e) {}

              @Override
              public void mouseDown(MouseEvent e) {

                    gameController.createCell(e.y/10,e.x/10,e.button);
                    pressed = e.button;

              }

              @Override
              public void mouseUp(MouseEvent e) {
```

```
                        pressed = 0;
                }
        }

        class MoveEditMode implements MouseMoveListener{
                @Override
                public void mouseMove(MouseEvent e) {
                                if(pressed>0)

        gameController.createCell(e.y/10,e.x/10,pressed);
                }

        }


        @Override
        public void update() {

                canvas.redraw();
        }

        public void changebackGroundColor(RGB rgb){
                canvas.setBackground(new Color(display, rgb));
        }

}
```

Il y plusieurs fonctionnalités dans notre GUI , parmi ses fonctionnalités :

- ouvrir ou enregistrer une configuration
- changer la couleur de fond ou bien des cellules
- mode éditeur avec la souris
- changer la vitesse pour la générations
- …

NB : la classe VueHelper utilise des méthodes static pour créer les différentes composants de l'interface graphique

Contrôleur

La classe GameController fournit les informations nécessaire pour le model à partir de la classe GameVue

Code source :

```
public class GameController {

	private GameOfLife gameOfLife;

	public GameController(GameOfLife gameOfLife) {
		this.gameOfLife =gameOfLife;
	}

	public GameOfLife getGameModel(){
		return gameOfLife;
	}


	public void createCell(int i,int j,int type){
		gameOfLife.createCell(i, j,type);
	}

	public void animate(){
		gameOfLife.nextGeneration();
	}

}
```

En résume:

- Le jeu de la vie est une automate cellulaire , et ce qui est magnifique c que a partir des règles simple John Conway a crée une révolutions dans l'univers des mathématiques
- L'api SWT est fourni par IDE eclipse , pour créer des applications professionnelle on n'utilise pas SWT tous seul , il existe ce qu'on appelle des application RCP(Rich Client Platform) qui sont basées sur l'api SWT et ils utilisent le noyau de IDE eclipse
- Le pattern MVC est très utiliser de nos jours , donc c'est une obligations d'étudier ce pattern ,car on le trouve dans Android, Symphony , PHP , Anguler JS , et plein d'autres langages
- L'implémentation de pattern MVC change d'un langage ç un autre mais le prince reste le même , en Java le pattern MVC est composé de deux d'autres patterns , Le pattern Observable/Observer et le pattern Stratégie .

Voila on a arrivé, c'est la fin.

Espérant que vous avez trouvé cet article utile, c'était une bonne expérience de savoir comment fonctionne le jeu de la vie , l'implémentation du MVC

Un grand merci à Monsieur Brixi-Nigassa et les étudiants de L2 Informatique .