INF 1404 – Java

Compte rendu de projet

Antoine STELLA

Lilian CIZERON

PEI L2

Introduction :

Dans le cadre de l'UE INF 1404, nous avons un projet à réaliser. Nous devons concevoir un programme en langage Java, capable de résoudre une énigme. Cette énigme est la résolution du parcours le plus cours d'un faisceau de laser, sur une carte constituée d'obstacle, comme dans le jeu *Mystère du Laser*, disponible sur Android. Nous devons donc concevoir une interface pour permettre à l'utilisateur de créer une carte avec des obstacles, de positionner un laser, ainsi qu'un algorithme capable de trouver comment le laser peut remplir le plus de cases, en en parcourant le moins possible, dans un temps le plus court possible.

Pour mener à bien ce projet, il nous a fallu poser des hypothèses. Tout d'abord, nous autorisons le faisceau laser à se chevaucher, c'est-à-dire qu'il peut passer deux fois au même endroit. Ensuite, nous considérons une carte de huit cases par huit. Cette taille pourra être modifiée ultérieurement aisément car nous allons réaliser le projet grâce à ce nombre à l'intérieur d'une variable.

Analyse du projet :

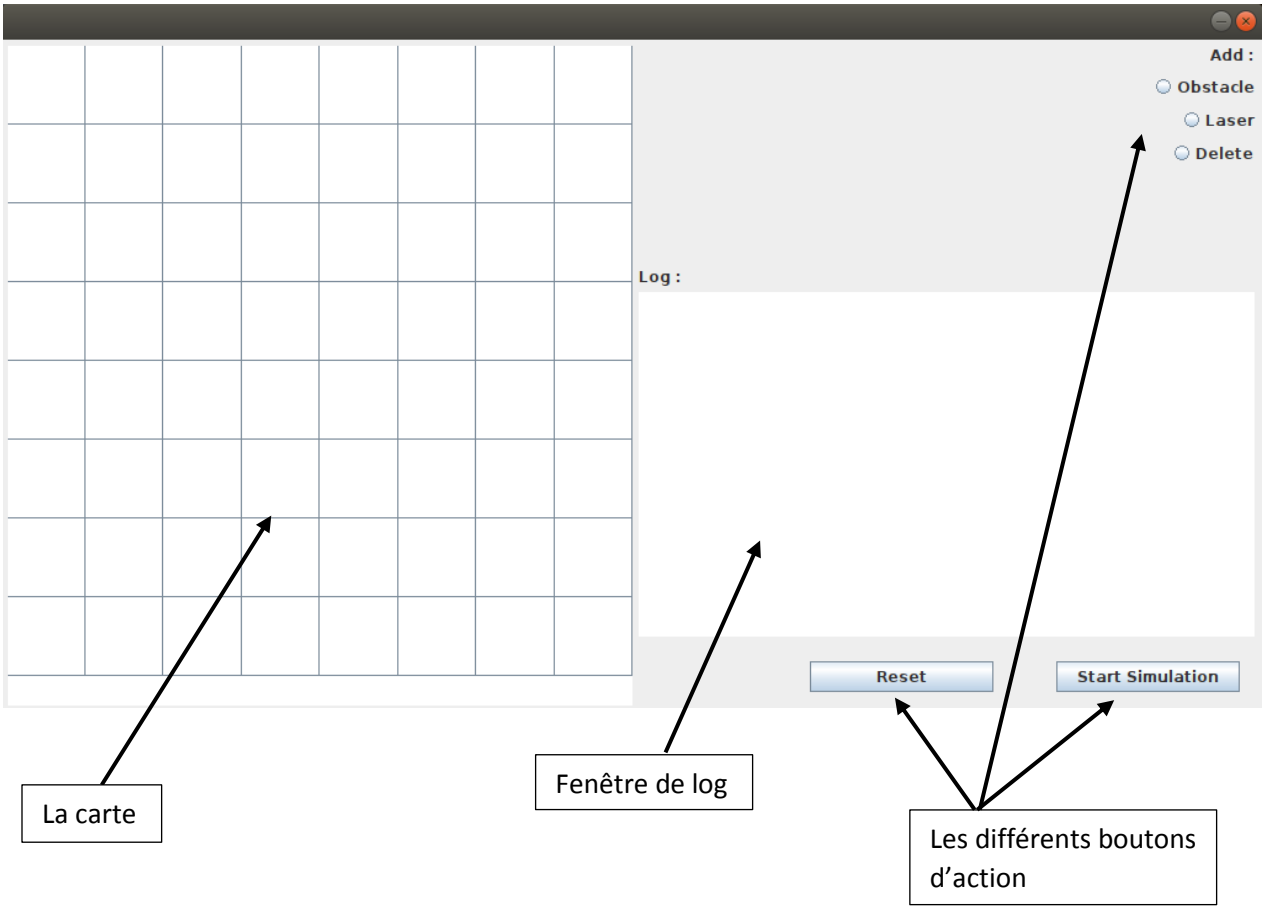
Conception du projet :

Nous avons réalisé ce projet à deux. Compte tenu du contexte, il nous a été impossible de pouvoir se rencontrer pour travailler

ensemble aisément. Nous avons donc décider de découper le travail en deux parties. Lilian s'est majoritairement penché sur la partie interface graphique, avec notamment la création de la carte, des objets du programme et du dessin de la carte. Antoine a travaillé sur la partie résolution du problème, avec notamment la recherche de la solution pour la carte créée par l'utilisateur, la conception de la pile, ainsi que l'optimisation du temps de recherche. Grâce à cette répartition du travail, nous avons pu travailler de manière productive et efficace.

Nous retrouvons donc deux principaux éléments dans ce projet : la partie Interface Homme Machine (IHM), et la partie algorithmique. La partie IHM se compose de deux parties : la partie de gauche permet l'affichage et la génération de la carte, et la partie droite permet de choisir l'objet à poser sur la carte, à lancer la recherche, ou à réinitialiser le programme.

Figure 1 : IHM de notre programme



Ainsi, si nous cochons la case « Obstacle » à droite, nous serons à même à poser un obstacle sur la carte de gauche. De même, si nous cochons la case « Laser », nous pourrons poser un laser, et si nous cochons la case « Delete », nous pouvons supprimer un objet de la carte.

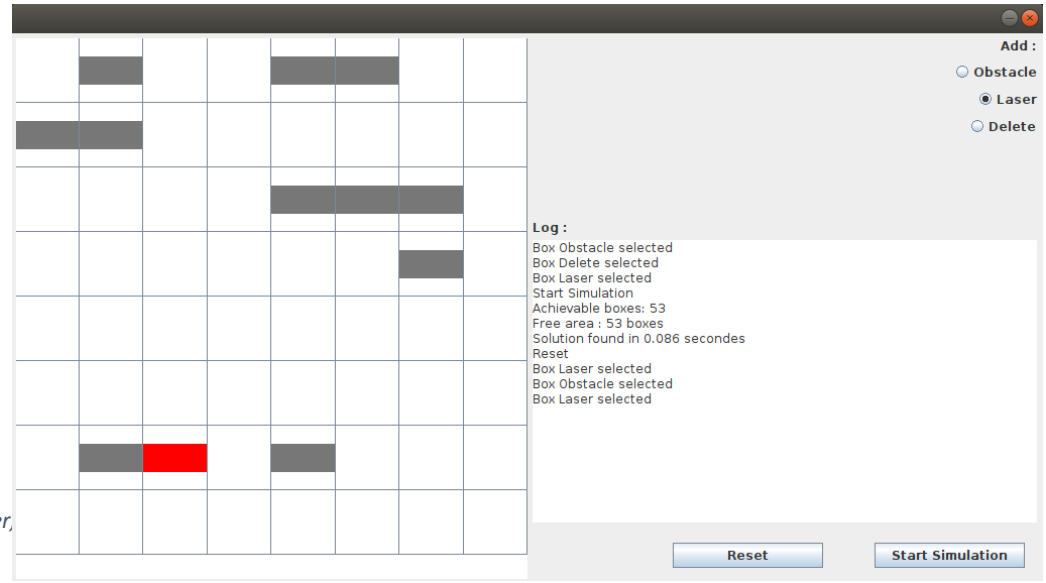


Figure 2 : Exemple de carte. En rouge le laser, en gris les obstacles

En outre, en cliquant sur le bouton « Reset », la carte va se vider, et nous pourrons refaire une simulation. Aussi, la recherche de la solution s'effectue lors de la pression du bouton « Start Simulation ». A partir de ce moment, l'algorithme va rechercher la meilleure solution au problème puis la dessiner.

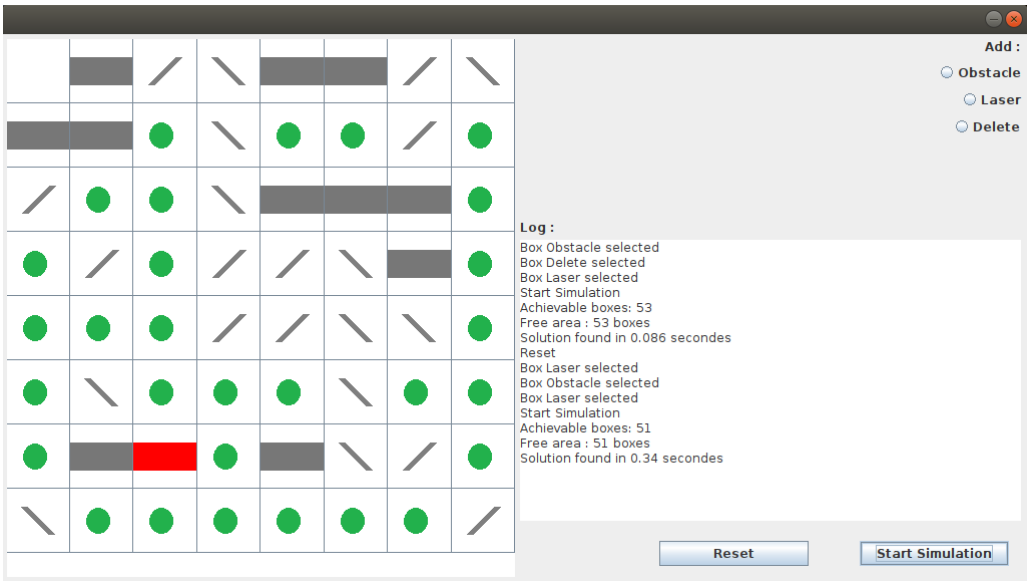


Figure 3 : Exemple de carte résolue. En rouge le laser, en gris les obstacles, en vert le faisceau du laser. En gris de côté : les miroirs

Le fonctionnement de l'algorithme :

Avant de se plonger dans l'explication détaillée du fonctionnement de notre algorithme, il est nécessaire de faire une étude mathématique du problème posé, à savoir « le faisceau parcourt un maximum de cases en un minimum de miroir ». Nous ne cherchons pas l'ensemble de solutions qui satisfassent le problème posé. En effet, en plus du peu d'intérêt, cela prendra un temps considérable. Le temps est par ailleurs un facteur très important pour notre algorithme, nous expliquerons pourquoi ensuite. Nous concevons donc un algorithme dont le but est de trouver une solution couvrant un maximum de cases puis en un minimum de miroirs.

Le piège dans la conception de l'algorithme est le temps. Nous avons à faire à un problème à complexité exponentielle, c'est-à-dire que plus nous ajoutons de cases, plus le temps de recherche d'une solution sera exponentiellement important. Nous avons des ordinateurs très puissants capable d'effectuer des milliards d'opérations par seconde, mais il ne faut pas sous-estimer la complexité exponentielle d'un problème.

Une des solutions pour répondre à ce problème consiste à tester une imposante quantité de combinaison de manière intelligente. Cette méthode étudiée en cours consiste à utiliser une pile pour effectuer des tests de chemin. Si un chemin n'aboutit pas sur la solution voulue, nous revenons en arrière pour tester autre chose : c'est le principe de fonctionnement de l'algorithme.

Détaillons le fonctionnement de la pile. Cette dernière contient des objets *déplacement*. Un *déplacement* contient 4 paramètres :

- Un choix précédent
- La position X précédente

- La position Y précédente
- La direction précédente

Nous pouvons en ajouter un au-dessus de la pile, le retirer, connaître la taille de la pile ainsi que de savoir quel est le *déplacement* au sommet. Ces actions sont suffisantes pour notre algorithme.

Comment remplir notre pile de *déplacement* ? Nous avons choisi de définir les choix de déplacement de la manière suivante.

Si le faisceau peut aller tout droit, il y va.

Sinon, si le faisceau peut aller à droite, il y va.

Sinon, si le faisceau peut aller à gauche, il y va.

Sinon, le faisceau annule son déplacement.

Si le faisceau ne peut ni aller tout droit, ni aller à droite, ni aller à gauche, alors il revient en arrière et suit l'ordre suivant. Par exemple, le faisceau va tout droit mais cela mène dans une impasse, alors il revient en arrière et l'action suivante est d'aller à droite. Hélas c'est encore une impasse, alors le faisceau revient en arrière et l'action suivante est d'aller à gauche. Et ainsi de suite, ces déplacements sont consignés par la pile.

Nous répétons cet ordre de décision tant que toutes les cases vides ne sont pas remplies par un faisceau. Et si le faisceau, n'ayant pu couvrir toutes les cases, revient en arrière jusqu'au laser, nous recommençons la simulation mais en cherchant à recouvrir toutes les cases moins une, cela jusqu'à ce qu'une solution soit obtenue.

Bien évidemment, nous avons protégé l'algorithme d'éventuels problèmes insolubles, comme un laser bloqué par un obstacle.

Mais nous avons trouvé le temps de calcul encore assez long dans un certains nombres de situation. Voici quelques exemples de situations où une optimisation algorithmique a permis de réduire considérablement le temps de calcul :

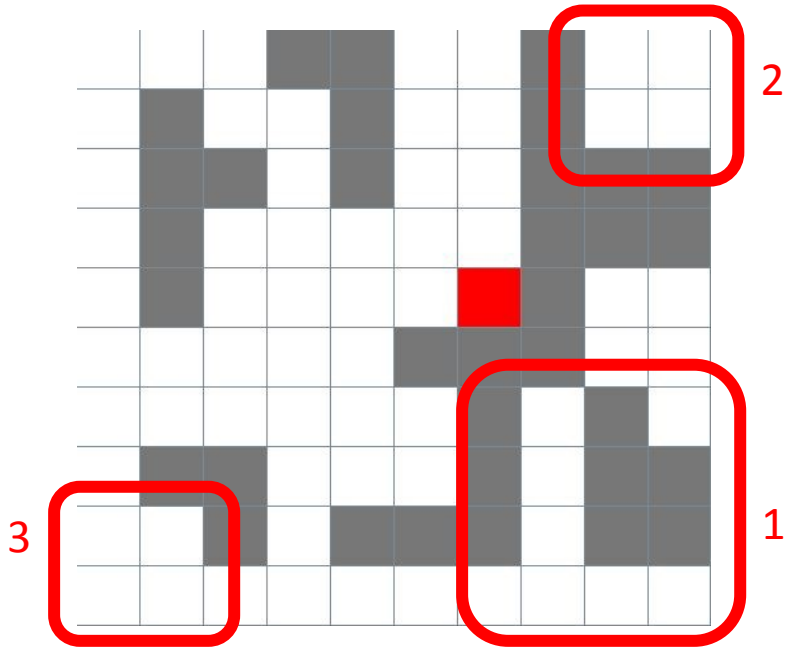


Figure 4: Exemple de carte à situations complexes

Cette carte contient trois zones pouvant réduire le nombre de tentatives de recherche de solution.

La zone 1 contient deux culs-de-sac. Nous cherchons à couvrir un maximum de cases, donc nous savons qu'en présence d'un cul-de-sac, le faisceau doit terminer dans ce dernier. Mais en présence de plusieurs culs-de-sac, que se passe-t-il ? Notre algorithme sait détecter un cul-de-sac, et nous savons que nous ne pouvons terminer que dans un sel cul-de-sac. Par conséquent, l'algorithme va automatiquement éliminer des cases à atteindre dans tous les culs-de-sac, sauf le plus grand. Sur l'exemple plus haut, il y a 65 cases d'air.

Mais, ayant éliminé le petit cul-de-sac de la zone 1, il ne cherchera à atteindre que 63 cases.

De plus, la zone 2 contient des cases complètement isolées du laser, donc totalement inaccessibles par le faisceau. De même, l'algorithme sait détecter ces cases et va les déduire des cases à atteindre. Il y a 9 cases isolées du laser, l'algorithme ne cherchera donc plus qu'à atteindre 54 cases.

Enfin, la zone 3 présente un cas de cases inaccessibles particulièrement bien caché. Les quatre cases de la zone sont bien accessibles physiquement par le faisceau, mais toutes ne le sont pas de manière logique. Quelque soit le chemin du laser, il y en aura une qui ne sera jamais atteinte. L'algorithme sait les trouver et les déduire du nombre de cases à atteindre. Ce nombre s'élève désormais à 53 cases.

Ainsi, dans cette situation nous sommes passés de 65 cases d'air à 53 cases atteignables. Cette diminution peut sembler peu importante, mais en réalité cela permet d'éviter à l'ordinateur de devoir chercher une solution recouvrant de 65 à 54 cases. Et cela représente un très grand nombre de solutions. Cette partie de l'algorithme a demandé beaucoup de temps et nous savons que ce temps de perdu à programmer sera gagné en utilisation, notamment sur des cartes de très grandes tailles. Nous rappelons que le temps d'exécution est un facteur important. Voici un comparatif de performances sur cette carte entre notre algorithme sans optimisation et avec :

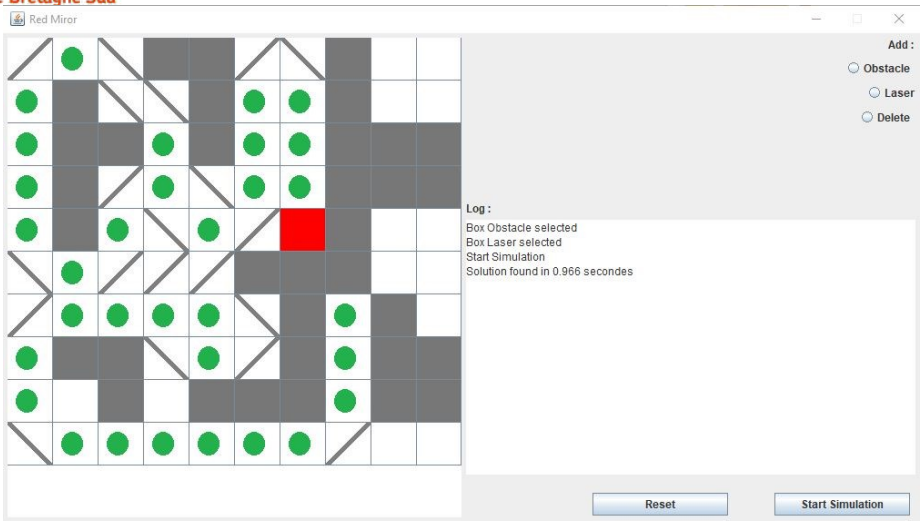


Figure 5: résolution sans optimisation

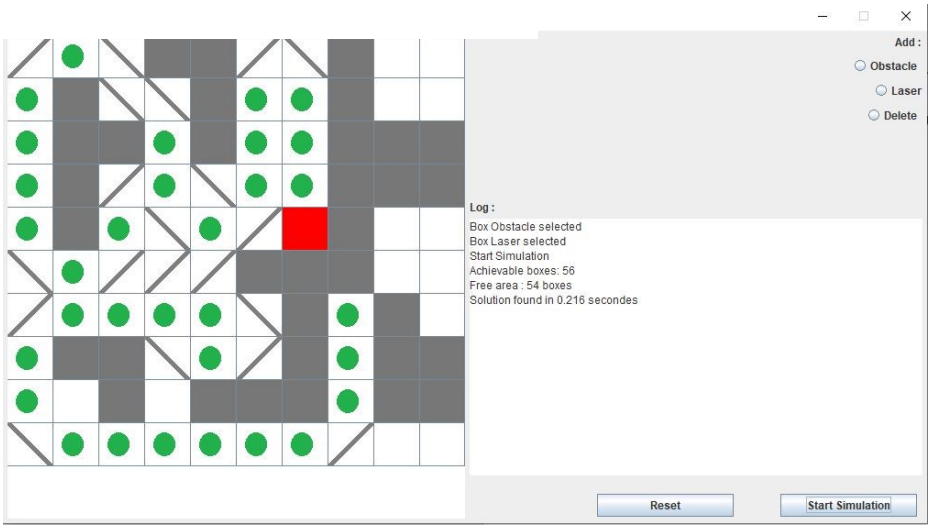


Figure 6: résolution avec optimisation

Nous sommes passés de 966 millisecondes à 216 millisecondes. Toutefois, ce gain n'est pas nécessairement notable, surtout dans les cas simples ne présentant aucune des situations évoquées.

En ce qui concerne la mémorisation du chemin parcouru, nous avons montré que c'était la pile qui enregistrait les *déplacements*. Pour enregistrer les obstacles, le laser, les faisceaux ainsi que les miroirs, nous avons créé un objet *map* qui contient un tableau de dimension n*n et toute une série de variables nécessaires au bon fonctionnement. Le tableau de cette *map* contient des entiers dont nous avons associés un élément graphique. Par exemple une case d'air

correspond à l'entier 0, un faisceau correspond à l'entier 3 et un obstacle correspond à l'entier 1. Ainsi il est très facile de manipuler ce tableau pour l'algorithme car les index correspondent aux coordonnées graphiques.

Voici comment à l'aide d'une pile nous pouvons mettre en place une solution algorithmique pour répondre à un problème mathématique simple d'apparence mais complexe dans la résolution. Cela est en partie permis par la grande puissance de calcul de nos ordinateurs personnels.

Conclusion :

Nous avons donc réussi à écrire un programme en java capable de résoudre un problème mathématique complexe : parcourir le maximum de case d'un plateau avec le minimum de miroir. En revanche, notre code a des limites. Pour créer correctement la carte, il faut être précis sur le positionnement des obstacles à l'aide de la souris. Notre code exclu aussi certaines possibilités, mais cela est nécessaire afin de pouvoir faire tourner le programme sur nos machines. Nous pourrions toujours optimiser le code afin de calculer le plus rapidement possible la solution.

Annexe – Code source :

Main.java :

```
package red_miror;

import java.lang.System;
//import java.util.Scanner;
//import java.util.Calendar;

public class Main {

    public static void main(String[] args) {

      //Variable
      //int taille = 10;
      //Pour les tests
      int taille = -1;
      try{
            taille = Integer.parseInt(args[0]);                    //Taille
de la carte n*n
            }
      catch(Exception e) {
            System.out.println("Please, type an integer.");
            System.exit(-1);
      }
        Map map = new Map(taille);
      //Instanciation de la carte


        //Scanner sc = new Scanner(System.in);                     //Pour
utiliser le touche par touche


        //Partie graphique
        int Xresolution = 1024, Yresolution = 576;
      //Résolution de la fenetre
        Interface frame = new Interface(taille, Xresolution, Yresolution,
map);
        frame.setSize(Xresolution, Yresolution);
        frame.setResizable(false);
        frame.setVisible(true);
        frame.setTitle("Red Miror");
     }
}
```

Deplacement.java :

```
package red_miror;

public class Deplacement {

	int choixPrecedent;
	int posPrecedentX;
	int posPrecedentY;
	int directionPrecedente;

	Deplacement(int x, int y, int choix, int direction) {
		choixPrecedent=choix;
		posPrecedentX=x;
		posPrecedentY=y;
		directionPrecedente=direction;
	}
}
```

Map.java :

```
package red_miror;

import javax.swing.ImageIcon;
import javax.swing.JTextArea;
import javax.swing.table.DefaultTableModel;

public class Map {

	/*
	 * Note importante:
	 * Pour eviter des erreurs de manipulations du tableau (out of
bounds, ...)
	 * Veuillez renseigner des coordonnees allant de 1 a ptaille. Les
indices
	 * a zero sont les murs de la piece, comme ceux a ptaille+1. Donc le
	 * premier element du tableau se situe aux coordonnees (1,1) et le
dernier
	 * element se situe au coordonnees (ptaille,ptaille)
	 * Pour ajouter des elements dans le tableau, uniquement passer par
	 * addObject(...)
	 */

	private int [][] piece;
	private int taille;
	private int positionLaserX=-1;
	private int positionLaserY=-1;
	private int surfaceMax;
	private int surfaceRealisable;
	private int nbrCroisement;
	private ImageIcon Miror_r = new
ImageIcon(Interface.class.getResource("/red_miror/Miror_r.png"));
```

```
	private ImageIcon Miror_l = new
ImageIcon(Interface.class.getResource("/red_miror/Miror_l.png"));
	private ImageIcon faisceau = new
ImageIcon(Interface.class.getResource("/red_miror/faisceau.png"));

	Map(int ptaille) {
		int i=0,j=0;
		taille=ptaille+2;
		piece = new int [taille][taille];
		for(j=0;j<taille;j++) {
			for(i=0;i<taille;i++) {
				if(j==0||j==(taille-1)||i==0||i==(taille-1)) {
					piece[i][j]=1;
				} else {
					piece [i][j]=0;
				}

			}
		}
		surfaceMax=ptaille*ptaille;
		surfaceRealisable=surfaceMax;
		nbrCroisement=0;
	}//fin constructeur

	//Obtenir la positionX du laser
	int getPositionLaserX() {
		return positionLaserX;
	}

	//Obtenir la positionY du laser
	int getPositionLaserY() {
		return positionLaserY;
	}

	//Definir la position du laser
	private void setPositionLaser(int x, int y) {
		positionLaserX=x;
		positionLaserY=y;
		piece[x][y]=2;
	}

	public boolean laserBloque() {
		return piece[positionLaserX][positionLaserY-1]==1;
	}

	//Determination du choix suivant en fonction du choix precedent, de
la position actuelle et de la direction
	int choixSuivant(int i, int j, int choixPrecedent, int direction) {
		//0 2
		//Monitoring
		//this.tracer();
		//System.out.println("\n");
		if(choixPrecedent==-2) {			//Premier deplacement
			//Tout droit ?
			if(direction==1) {if(piece[i+1][j]==0) return 1;
if(piece[i+1][j]==3) {if(piece[i+2][j]==0) return 1;}}
			if(direction==2) {if(piece[i][j-1]==0) return 1;
if(piece[i][j-1]==3) {if(piece[i][j-2]==0) return 1;}}
```

```
                        if(direction==3) {if(piece[i-1][j]==0) return 1;
if(piece[i-1][j]==3) {if(piece[i-2][j]==0) return 1;}}
                        if(direction==4) {if(piece[i][j+1]==0) return 1;
if(piece[i][j+1]==3) {if(piece[i][j-2]==0) return 1;}}
                  }
                  if(choixPrecedent==0) {
                        //Tout droit ?
                        if(direction==1) {if(piece[i+1][j]==0) return 1;
if(piece[i+1][j]==3) {if(piece[i+2][j]==0) return 1;}}
                        if(direction==2) {if(piece[i][j-1]==0) return 1;
if(piece[i][j-1]==3) {if(piece[i][j-2]==0) return 1;}}
                        if(direction==3) {if(piece[i-1][j]==0) return 1;
if(piece[i-1][j]==3) {if(piece[i-2][j]==0) return 1;}}
                        if(direction==4) {if(piece[i][j+1]==0) return 1;
if(piece[i][j+1]==3) {if(piece[i][j-2]==0) return 1;}}
                        //Droite ?
                        if(direction==1) {if(piece[i][j+1]==0) return 2;
if(piece[i][j+1]==3 ) {if(piece[i][j+2]==0) return 2;}}
                        if(direction==2) {if(piece[i+1][j]==0) return 2;
if(piece[i+1][j]==3 ) {if(piece[i+2][j]==0) return 2;}}
                        if(direction==3) {if(piece[i][j-1]==0) return 2;
if(piece[i][j-1]==3 ) {if(piece[i][j-2]==0) return 2;}}
                        if(direction==4) {if(piece[i-1][j]==0) return 2;
if(piece[i-1][j]==3 ) {if(piece[i-2][j]==0) return 2;}}
                        //Gauche ?
                        if(direction==1) {if(piece[i][j-1]==0) return 3;
if(piece[i][j-1]==3 ) {if(piece[i][j-2]==0) return 3;}}
                        if(direction==2) {if(piece[i-1][j]==0) return 3;
if(piece[i-1][j]==3 ) {if(piece[i-2][j]==0) return 3;}}
                        if(direction==3) {if(piece[i][j+1]==0) return 3;
if(piece[i][j+1]==3 ) {if(piece[i][j+2]==0) return 3;}}
                        if(direction==4) {if(piece[i+1][j]==0) return 3;
if(piece[i+1][j]==3 ) {if(piece[i+2][j]==0) return 3;}}
                  }
                  if(choixPrecedent==1) {
                        //Droite ?
                        if(direction==1) {if(piece[i][j+1]==0) return 2;
if(piece[i][j+1]==3 ) {if(piece[i][j+2]==0) return 2;}}
                        if(direction==2) {if(piece[i+1][j]==0) return 2;
if(piece[i+1][j]==3 ) {if(piece[i+2][j]==0) return 2;}}
                        if(direction==3) {if(piece[i][j-1]==0) return 2;
if(piece[i][j-1]==3 ) {if(piece[i][j-2]==0) return 2;}}
                        if(direction==4) {if(piece[i-1][j]==0) return 2;
if(piece[i-1][j]==3 ) {if(piece[i-2][j]==0) return 2;}}
                        //Gauche ?
                        if(direction==1) {if(piece[i][j-1]==0) return 3;
if(piece[i][j-1]==3 ) {if(piece[i][j-2]==0) return 3;}}
                        if(direction==2) {if(piece[i-1][j]==0) return 3;
if(piece[i-1][j]==3 ) {if(piece[i-2][j]==0) return 3;}}
                        if(direction==3) {if(piece[i][j+1]==0) return 3;
if(piece[i][j+1]==3 ) {if(piece[i][j+2]==0) return 3;}}
                        if(direction==4) {if(piece[i+1][j]==0) return 3;
if(piece[i+1][j]==3 ) {if(piece[i+2][j]==0) return 3;}}
                  }
                  if(choixPrecedent==2) {
                        //Gauche ?
                        if(direction==1) {if(piece[i][j-1]==0) return 3;
if(piece[i][j-1]==3 ) {if(piece[i][j-2]==0) return 3;}}
```

```
                        if(direction==2) {if(piece[i-1][j]==0) return 3;
if(piece[i-1][j]==3 ) {if(piece[i-2][j]==0) return 3;}}
                        if(direction==3) {if(piece[i][j+1]==0) return 3;
if(piece[i][j+1]==3 ) {if(piece[i][j+2]==0) return 3;}}
                        if(direction==4) {if(piece[i+1][j]==0) return 3;
if(piece[i+1][j]==3 ) {if(piece[i+2][j]==0) return 3;}}
                  }
                  if(choixPrecedent==3) return -1;
                  return -1;
            }

            //Ajoute le type d'un objet dans le tableau: 0=air 1=obstacle 2=laser
            public void addObject(int type, int positionY, int positionX) {
                  positionY++; positionX++;
            //Correction du d?calage du tableau dans interface.java
                  if(positionX<taille && positionY<taille && positionX>0 &&
positionY>0) {          //Verification out of bounds
                        System.out.println(type + " en
X="+positionX+":Y="+positionY);
            //Monitoring
                        piece[positionX][positionY]=type;
                        if(type==2) {
                              this.setPositionLaser(positionX, positionY);
                        }
                  }
                  this.majSurfaceMax();
            }

            //Est ce que le tableau est plein ?
            public boolean plein(int taillePile) {
                  //System.out.println(taillePile-nbrCroisement);
            //Monitoring
                  return (taillePile-nbrCroisement)==surfaceRealisable;
            }

            //Place un objet a une position donnee
            public void set(int i, int j, boolean pSlash, int pNouveauChoix) {
                  if(pNouveauChoix==1) {        //Si tout droit
                        if(piece[i][j]==3) {    //Si il y a deja un faisceau alors
croisement
                              piece[i][j]=6;
                              nbrCroisement++;
                        } else {
                              piece[i][j]=3;
                        }
                  } else {
                        if(pSlash==true) {            //Si miroir en '/'
                              piece[i][j]=4;
                        }
                        if(pSlash==false) {           //Si miroir en '\'
                              piece[i][j]=5;
                        }
                  }
            }

            //Place de l'air a une position donnee
            public void reset(int i, int j) {
                  if(piece[i][j]==6) {                //Si retour sur croisement
```

```
                        piece[i][j]=3;                          //On replace un
faisceau
                        nbrCroisement--;
                  } else {
                        piece[i][j]=0;
                  }
                  piece[positionLaserX][positionLaserY]=2;
            }

            //Affiche dans la console le tableau de la map
            public void tracer(DefaultTableModel modele) {
                  int i=0, j=0;
                  for(j=1;j<taille-1;j++) {
                        for(i=1;i<taille-1;i++) {
                              System.out.print(piece[i][j]);
            //Monitoring
                              if (piece[i][j]==3 || piece[i][j]==6) {
                                    modele.setValueAt(faisceau, j-1, i-1);
                              }
                              if (piece[i][j]==4) {
                                    modele.setValueAt(Miror_r, j-1, i-1);
                              }
                              if (piece[i][j]==5) {
                                    modele.setValueAt(Miror_l, j-1, i-1);
                              }
                              modele.fireTableDataChanged();
                        }
                        System.out.println("");                  //Monitoring
                  }
            }

            //Met  jour la valeur de surface libre maximum sur la map
            //Le laser n est pas compte comme surface libre
            private void majSurfaceMax() {
                  surfaceMax=(taille-2)*(taille-2);
                  int i=0, j=0;
                  for(j=1;j<(taille-1);j++) {
                        for(i=1;(i<taille-1);i++) {
                              if(piece[i][j]!=0) {
                                    surfaceMax--;
                              }
                        }
                  }
                  surfaceRealisable=surfaceMax;
            }

            //Met a jour la valeur de surface realisable au maximum par le laser
            // => elimination des cases isolees du laser (inatteignable)
            // => elimination des cases cul-de-sac sauf le + grand qui sera la
fin
            //du parcours du faisceau
            public void majSurfaceRealisable(JTextArea textLog) {
                  int i=0, j=0, ii=0, jj=0, k=-1, l=0;
                  int dir=0, dirSuivante=0;
                  int longueur=0, tailleMax=0, casesSept=1;
                  //cases a 7: cases atteignables par le laser, non isolees,
comptees par la variable casesSept
```

```
                //Recherche de cases isolees
                if(piece[this.getPositionLaserX()][this.getPositionLaserY()-
1]==0) {          //Si le debut du faisceau est bien de l air
                      piece[this.getPositionLaserX()][this.getPositionLaserY()-
1]=7;
                      while(k!=0) {          //Tant qu il y a de nouvelles
cases atteignables de decouvertes
                            k=0;
                            for(j=1;j<taille-1;j++) {
                                  for(i=1;i<taille-1;i++) {
                                        if(piece[i][j]==0 && piece[i+1][j]==7)
{piece[i][j]=7; k++; casesSept++;}
                                        if(piece[i][j]==0 && piece[i][j-1]==7)
{piece[i][j]=7; k++; casesSept++;}
                                        if(piece[i][j]==0 && piece[i-1][j]==7)
{piece[i][j]=7; k++; casesSept++;}
                                        if(piece[i][j]==0 && piece[i][j+1]==7)
{piece[i][j]=7; k++; casesSept++;}
                                  }//fin for i
                            }//fin for j
                      }//fin while
                      surfaceRealisable=casesSept;
                }//fin if
                textLog.append("Achievable boxes: "+casesSept+"\n");

                //Recherche des culs-de-sac
                for(j=1;j<taille-1;j++) {
                      for(i=1;i<taille-1;i++) {
                            k=0; dir=0; dirSuivante=0;
                            //Comptage des obstacles autour de (i,j)
                            if(piece[i+1][j]==1) {k++;} else {dir=1;}
                            if(piece[i][j-1]==1) {k++;} else {dir=2;}
                            if(piece[i-1][j]==1) {k++;} else {dir=3;}
                            if(piece[i][j+1]==1) {k++;} else {dir=4;}
                            //Si 3 obstacles alors cul-de-sac
                            if(k==3 && piece[i][j]==7) {
                                  //System.out.println("cul-de-sac:"+i+","+j);
                      //Monitoring
                                  ii=i; jj=j;
                                  longueur=0;
                                  do {
                                        longueur++;
                                        surfaceRealisable--;
                                        if(dir==1) ii++;
                                        if(dir==2) jj--;
                                        if(dir==3) ii--;
                                        if(dir==4) jj++;
                                        l=0;
                                        //Comptage des obstacles autour de
(ii,jj)
                                        if(piece[ii+1][jj]==1) l++; else
if(dir!=3) dirSuivante=1;
                                        if(piece[ii][jj-1]==1) l++; else
if(dir!=4) dirSuivante=2;
                                        if(piece[ii-1][jj]==1) l++; else
if(dir!=1) dirSuivante=3;
                                        if(piece[ii][jj+1]==1) l++; else
if(dir!=2) dirSuivante=4;
                                        if(l==2) {
```

```
                        dir=dirSuivante;
                    }
                }while(l==2);           //Tant que dans le
cul-de-sac
                //Enregistrement du record de taille de cul-
de-sac
                if(longueur>tailleMax) tailleMax=longueur;
            }
        }//fin for i
    }//fin for j
    //On remet les cases a 7 a 0
    for(j=1;j<taille-1;j++) {
        for(i=1;i<taille-1;i++) {
            if(piece[i][j]==7) {
                piece[i][j]=0;
            }
        }
    }
    surfaceRealisable=surfaceRealisable+tailleMax;
    textLog.append("Free area : "+surfaceRealisable+" boxes\n");
        //Monitoring
}

//Retire les faisceaux laser et les miroirs
public void resetBeam() {
    int i=0,j=0;
    for(j=1;j<taille-1;j++) {
        for(i=1;i<taille-1;i++) {
            if(piece[i][j]==3 || piece[i][j]==4 ||
piece[i][j]==5 || piece[i][j]==6) {
                piece[i][j]=0;
            }
        }
    }
}

//Retire tous les objets de la map, r?initialise toutes les valeurs
de simulation:
//positionLaserX, positionLaserY, surfaceMax, surfaceRealisable,
nbrCroisement
public void resetAll() {
    positionLaserX=-1;
    positionLaserY=-1;
    surfaceMax=(taille-2)*(taille-2);
    surfaceRealisable=surfaceMax;
    nbrCroisement=0;
    int i=0,j=0;
    for(j=1;j<taille-1;j++) {
        for(i=1;i<taille-1;i++) {
            piece[i][j]=0;
        }
    }
}
}
```

Pile.java :

```
package red_miror;

import java.util.Vector;

public class Pile <T>{
	Vector <T> table;

	Pile (){
		table = new Vector<T>();
	}

	void Empile (T x){
		table.add(table.size(), x);
	}

	T SommetPile(){
		return table.elementAt(table.size()-1);
	}

	void Depile(){
		table.remove(table.size()-1);
	}

	int Taille() {
		return table.size();
	}

}
```

Interface.java :

```
package red_miror;

import java.awt.event.*;                                    //Importe les
libraries nécessaires au fonctionnement du logiciel
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.ImageIcon;
import java.awt.GridLayout;
import javax.swing.JButton;
import javax.swing.JRadioButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import java.awt.Component;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumn;

import java.util.Calendar;
```

```
public class Interface extends JFrame {

	private static final long serialVersionUID = 1L;
	private JPanel contentPane;

	private int selection = -1;
					//Variable afin de savoir quelle type
d'objet on veut poser sur la map
	private JTable table_1;
	private boolean obsOK = false;
		//Savoir si le laser est déjà posé
	private boolean slash = false;
		//Type de miroir entre '/'=true et '\'=false
	private int direction=2;
		//Direction emprunt?e par le laser 1=E 2=N 3=O 4=S
	private int choixPrecedent=0, nouveauChoix=0, numeroDeplacement=1,
ii=0, jj=0, i=0, j=0, compensation=0;
	private Calendar calDebut;
	private	Calendar calFin;
	private Pile <Deplacement> P = new Pile<Deplacement>();
	//Creation de la pile de deplacement
	private Deplacement deplacementPrecedent;
	private TableColumn col = new TableColumn();

	public Interface(int taille, int Xresolution, int Yresolution, Map
map) {
	//Constructeur de notre interface

		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
				//Création de la fenêtre de base
		setBounds(100, 100, 450, 300);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
		setContentPane(contentPane);
		contentPane.setLayout(new GridLayout(1,2));

		ImageIcon vide = new
ImageIcon(Interface.class.getResource("/red_miror/white_solid.png"));
		//Création des objets pour l'affichage dans le tableau
		ImageIcon obstacle = new
ImageIcon(Interface.class.getResource("/red_miror/grey_solid.png"));
		ImageIcon laser = new
ImageIcon(Interface.class.getResource("/red_miror/red_solid.png"));

		Object[][] donnees = new Object[taille][taille];
								//Création
du tableau vide
		String[] entetes = new String[taille];
		for(i=0;i<taille;i++) {
			for(j=0;j<taille;j++) {
				donnees[i][j]=vide;
			}
			entetes[i]="0";
		}

	DefaultTableModel modele = new DefaultTableModel(donnees, entetes);
		table_1 = new JTable(modele)
```

```
		{
			private static final long serialVersionUID = -
8913506703926579066L;

			/*détection automatique des types de données
			 de toutes les colonnes
			 */
			@SuppressWarnings({ "unchecked", "rawtypes" })
			public Class getColumnClass(int colonne)
			{
			    return getValueAt(0, colonne).getClass();
			}

		   public boolean isCellEditable(int x, int y) {
			return false;
			}
		};
		table_1.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
		table_1.setRowHeight(Yresolution/taille);
		for (i=0;i<taille;i++) {
		    col = table_1.getColumnModel().getColumn(i);
		    col.setWidth(Xresolution/taille);
		}
		col = table_1.getColumnModel().getColumn(0);
		contentPane.add(table_1);

			JPanel panel_1 = new JPanel();
					//Création de l'emplacement de nos
boutons
			panel_1.setAlignmentX(Component.RIGHT_ALIGNMENT);
			contentPane.add(panel_1);
			panel_1.setLayout(null);

			JButton btnStartSimulation = new JButton("Start Simulation");
			//Création du bouton pour lancer la simulation
			btnStartSimulation.setBounds(Xresolution-680, Yresolution-76,
149, 25);
			panel_1.add(btnStartSimulation);

			JButton btnReset = new JButton("Reset");
//Création du bouton Reset
		btnReset.setBounds(Xresolution-880, Yresolution-76, 149, 25);
		panel_1.add(btnReset);

			JRadioButton rdbtnObstacle = new JRadioButton("Obstacle");
			//Création de la sélection 1 : l'obstacle
			rdbtnObstacle.setBounds(Xresolution-603, Yresolution-554, 88,
23);
			panel_1.add(rdbtnObstacle);

			JRadioButton rdbtnLaser = new JRadioButton("Laser");
				//Création de la sélection 2 : le laser
			rdbtnLaser.setBounds(Xresolution-580, Yresolution-527, 65, 23);
			panel_1.add(rdbtnLaser);

			JRadioButton rdbtnDelete = new JRadioButton("Delete");
				//Création de la sélection 3 : supprimer
			rdbtnDelete.setBounds(Xresolution-588, Yresolution-500, 80,
23);
```

```
		panel_1.add(rdbtnDelete);

		JLabel lblAdd = new JLabel("Add :");
				//Création de l'indication de sélection de
l'objet à ajouter
		lblAdd.setBounds(Xresolution-555, 0, 42, 15);
		panel_1.add(lblAdd);

		JLabel lblLog = new JLabel("Log :");
				//Création de l'indication de log
		lblLog.setBounds(5, 180, 42, 15);
		panel_1.add(lblLog);
		JTextArea textLog = new JTextArea();
				//Création de la fenêtre de log
		textLog.setBounds(5, 200, 500, 280);
		panel_1.add(textLog);
		textLog.setEditable(false);

		JLabel lblObstacle = new JLabel("");
				//Création de l'image de l'obstacle
		lblObstacle.setEnabled(false);
		lblObstacle.setVisible(false);
		lblObstacle.setIcon(obstacle);
		lblObstacle.setBounds(0, 0, col.getWidth(),
table_1.getRowHeight(0));
		table_1.add(lblObstacle);

		JLabel lblLaser = new JLabel("");
				//Création de l'image du laser
		lblLaser.setEnabled(false);
		lblLaser.setVisible(false);
		lblLaser.setIcon(laser);
		lblLaser.setBounds(0, 0, col.getWidth(),
table_1.getRowHeight(0));
		table_1.add(lblLaser);


		rdbtnObstacle.addItemListener(new ItemListener() {
		//Bloc permettant de suivre les actions de la sélection
Obstacle
			public void itemStateChanged(ItemEvent e) {
				if (e.getStateChange() == ItemEvent.SELECTED) {
					selection = 1;
					rdbtnLaser.setSelected(false);
					rdbtnDelete.setSelected(false);

					textLog.append("Box Obstacle selected\n");
				}
				if (e.getStateChange() == ItemEvent.DESELECTED &&
selection == 1) {
					rdbtnObstacle.setSelected(true);
				}
			}

		});

		rdbtnLaser.addItemListener(new ItemListener() {
		//Bloc permettant de suivre les actions de la sélection Laser
			public void itemStateChanged(ItemEvent e) {
```

```
                        if (e.getStateChange() == ItemEvent.SELECTED) {
                              selection = 2;
                              rdbtnObstacle.setSelected(false);
                              rdbtnDelete.setSelected(false);

                              textLog.append("Box Laser selected\n");
                        }
                        if (e.getStateChange() == ItemEvent.DESELECTED &&
selection == 2) {
                              rdbtnLaser.setSelected(true);
                        }
                  }

            });

            rdbtnDelete.addItemListener(new ItemListener() {
                  //Bloc permettant de suivre les actions de la sélection
Delete
                  public void itemStateChanged(ItemEvent e) {
                        if (e.getStateChange() == ItemEvent.SELECTED) {
                              selection = 0;
                              rdbtnObstacle.setSelected(false);
                              rdbtnLaser.setSelected(false);
                              textLog.append("Box Delete selected\n");
                        }
                        if (e.getStateChange() == ItemEvent.DESELECTED &&
selection == 0) {
                              rdbtnDelete.setSelected(true);
                        }
                  }

            });

            btnReset.addActionListener(new ActionListener() {
      //Bloc permettant de suivre les actions du bouton Reset
                  int i = 0,j = 0;
                  public void actionPerformed(ActionEvent e) {
                        textLog.append("Reset\n");
                        map.resetAll();
                        for(i=0; i<taille; i++) {
                            for(j=0; j<taille; j++) {
                                  modele.setValueAt(vide, i, j);
                                  }
                        }
                        obsOK=false;
                        modele.fireTableDataChanged();
                  }
            });

            btnStartSimulation.addActionListener(new ActionListener() {
            //Bloc permettant de suivre les actions du bouton Start
Simulation
                  JOptionPane box;
                  @SuppressWarnings("static-access")
                  public void actionPerformed(ActionEvent e) {
                        if (obsOK==true) {
                              textLog.append("Start Simulation\n");
                              selection=-1;
                              rdbtnObstacle.setSelected(false);
```

```
rdbtnLaser.setSelected(false);
rdbtnDelete.setSelected(false);
lblLaser.setEnabled(false);
lblLaser.setVisible(false);
lblObstacle.setEnabled(false);
lblObstacle.setVisible(false);
modele.fireTableDataChanged();

//Depart de la simulation
map.tracer(modele); System.out.println("");
                                    //Monitoring
//System.out.println("Appuyez sur une touche
pour lancer la r?solution.");      //Monitoring
//sc.nextLine();

      //Monitoring
calDebut=Calendar.getInstance();
map.majSurfaceRealisable(textLog);
                  //Mise a jour de la surface realisable (voir map.java)
if(!map.laserBloque()) {
      do {
            slash=false;
          direction=2;
          choixPrecedent=-2;
          nouveauChoix=0;
          numeroDeplacement=1;
          ii=0; jj=0;
          int i=map.getPositionLaserX();
          int j=map.getPositionLaserY();
          map.resetBeam();

          //Si la pile n'est pas vide, on
retire toutes les assiettes
          while(P.Taille()!=0) {
                P.Depile();
          }

      while(!map.plein(P.Taille()+compensation)) {          //Tant que la
map n'est pas pleine

      nouveauChoix=map.choixSuivant(i,j,choixPrecedent,direction);
                                          if(nouveauChoix!=-1) {
      //Si le nouveau choix n est pas une impasse
                                              P.Empile(new
Deplacement(i,j,nouveauChoix,direction));      //ajout d une assiette a la
pile
                                              ii=i; jj=j;
                                              if(nouveauChoix==1) {
            //Si tout droit

if(direction==1){i++;}
                                                  if(direction==2){j--
;}
                                                  if(direction==3){i--
;}

if(direction==4){j++;}
                                              }
```

```
        //Si droite via miroir
slash=false;}
slash=true;}
slash=false;}
slash=true;}
        //Changement de direction
{direction=4;}          //modulo

        //Si gauche via miroir
slash=true;}
slash=false;}
slash=true;}
slash=false;}
        //Changement de direction
{direction=1;}          //modulo


deplacement laser effectu

        map.set(i,j,slash,nouveauChoix);

        map.set(ii,jj,slash,nouveauChoix);

                        //Impasse donc retour arriere
        deplacementPrecedent=P.SommetPile();      //Recuperation du dernier
deplacement au sommet de la pile
                    //Retire la derniere assiette
                    //Retire le deplacement laser
        i=deplacementPrecedent.posPrecedentX;
        j=deplacementPrecedent.posPrecedentY;
```

```
                if(nouveauChoix==2) {
                    if(direction==1){j++;
                    if(direction==2){i++;
                    if(direction==3){j--;
                    if(direction==4){i--;
                    direction--;
                    if(direction==0)
                }
                if(nouveauChoix==3) {
                  if(direction==1){j--;
                    if(direction==2){i--;
                    if(direction==3){j++;
                    if(direction==4){i++;
                    direction++;
                    if(direction==5)
                }
                numeroDeplacement++;
                //Place sur la map le
                if(nouveauChoix==1) {
                } else {
                  map.set(i,j,slash,1);
                }
                choixPrecedent=0;
            }
            else if(numeroDeplacement!=1) {
                P.Depile();
                map.reset(i,j);
```

```
		direction=deplacementPrecedent.directionPrecedente;

		choixPrecedent=deplacementPrecedent.choixPrecedent;
                                                numeroDeplacement--;
                                            }
                                            else {
		//pas de solution
                                                break;
                                            }
                                            //sc.nextLine();
		//Monitoring: une tape  chaque touche
                                            //map.tracer();
		//Monitoring
                                        }//fin while
                                            if(nouveauChoix==-1 &&
numeroDeplacement==1)
                                            compensation++;

		System.out.println("Compensation="+compensation);
		//Monitoring
                                    }while(nouveauChoix==-1 &&
numeroDeplacement==1);//fin do while
                                } else {
                                    box = new JOptionPane();
                                    box.showMessageDialog(null, "Laser
obstructed, make it free", "Warning", JOptionPane.WARNING_MESSAGE);
                                }
                            map.tracer(modele);
                            calFin = Calendar.getInstance();
                            float tempsCalcul;
                            tempsCalcul =
(calFin.get(Calendar.HOUR_OF_DAY)*60*60*1000+calFin.get(Calendar.MINUTE)*60
*1000+calFin.get(Calendar.SECOND)*1000+calFin.get(Calendar.MILLISECOND))-
(calDebut.get(Calendar.HOUR_OF_DAY)*60*60*1000+calDebut.get(Calendar.MINUTE
)*60*1000+calDebut.get(Calendar.SECOND)*1000+calDebut.get(Calendar.MILLISEC
OND));
                            tempsCalcul=tempsCalcul/1000f;
                            textLog.append("Solution found in "+tempsCalcul+"
secondes\n");
                        }
                            else if(obsOK==false) {
                                box = new JOptionPane();
                                box.showMessageDialog(null, "You have to put
a laser on the map.", "Warning", JOptionPane.WARNING_MESSAGE);
                            }
                    }
                });


                table_1.addMouseMotionListener(new MouseMotionListener() {
                    //Bloc permettant de suivre les déplacements de la souris

                    public void mouseDragged(MouseEvent arg0) {
                        //Mouvement + clic
                        System.out.println("Log : Mouse dragged");
                    }
```

```
		public void mouseMoved(MouseEvent arg0) {
			//Gestion dynamique de l'affichage de l'objet sur le pointeur
			System.out.println("Log : Mouse moved : X = "+arg0.getX()+" ; Y = "+arg0.getY());
			lblLaser.setBounds(arg0.getX()-20, arg0.getY()-10, 33, 33);
			lblObstacle.setBounds(arg0.getX()-20, arg0.getY()-10, 33, 33);
			modele.fireTableDataChanged();

			if (arg0.getX()<504 && selection==1) {

				lblObstacle.setEnabled(true);
				lblObstacle.setVisible(true);
				lblLaser.setEnabled(false);
				lblLaser.setVisible(false);
			}
			else if (arg0.getX()<504 && selection==2){

				lblObstacle.setEnabled(false);
				lblObstacle.setVisible(false);
				lblLaser.setEnabled(true);
				lblLaser.setVisible(true);
			}
			else if (arg0.getX()>=504 || selection==0){
				lblObstacle.setEnabled(false);
				lblObstacle.setVisible(false);
				lblLaser.setEnabled(false);
				lblLaser.setVisible(false);
			}
		}


	});


	table_1.addMouseListener(new MouseAdapter() {  //Ajout d'un obstacle dans la case correspondante du tableau
		@SuppressWarnings("static-access")
		public void mouseClicked(MouseEvent e) {

			JOptionPane box;

			if (selection == 0) {
	//On supprime ce qui était placé
				try{
					map.addObject(selection, this.positionTableY(lblObstacle.getY()), this.positionTableX(lblObstacle.getX()));
					if (modele.getValueAt(this.positionTableY(lblObstacle.getY()), this.positionTableX(lblObstacle.getX())) == laser){
						obsOK = false;
					}
					modele.setValueAt(vide, this.positionTableY(lblObstacle.getY()), this.positionTableX(lblObstacle.getX()));
				}
```

```
                        catch(Exception j) {
                              box = new JOptionPane();
                              box.showMessageDialog(null, "You clicked
outside of the table !", "Warning", JOptionPane.WARNING_MESSAGE);
                        }
                              modele.fireTableDataChanged();
                  }

                  if (obsOK == true && selection == 2) {
                        box = new JOptionPane();
                        box.showMessageDialog(null, "You already have
a Laser, delete it first.", "Information",
JOptionPane.INFORMATION_MESSAGE);
                  }

                  if (selection == 2 && obsOK == false){
                              //On place un laser
                        try{
                              map.addObject(selection,
this.positionTableY(lblLaser.getY()),
this.positionTableX(lblLaser.getX()));
                              modele.setValueAt(laser,
this.positionTableY(lblLaser.getY()),
this.positionTableX(lblLaser.getX()));
                              obsOK = true;
                        }
                        catch(Exception j) {
                              box = new JOptionPane();
                              box.showMessageDialog(null, "You clicked
outside of the table !", "Warning", JOptionPane.WARNING_MESSAGE);
                        }
                        modele.fireTableDataChanged();
                  }

                  if (selection == 1){

                              //On place un obstacle
                        try{
                              if
(modele.getValueAt(this.positionTableY(lblObstacle.getY()),
this.positionTableX(lblObstacle.getX())) == laser){
                                    obsOK = false;
                              }
                              map.addObject(selection,
this.positionTableY(lblObstacle.getY()),
this.positionTableX(lblObstacle.getX()));
                              modele.setValueAt(obstacle,
this.positionTableY(lblObstacle.getY()),
this.positionTableX(lblObstacle.getX()));
                        }
                        catch (Exception j) {
                              box = new JOptionPane();
                              box.showMessageDialog(null, "You clicked
outside of the table !", "Warning", JOptionPane.WARNING_MESSAGE);
                        }
                        modele.fireTableDataChanged();
                        }
                  }
```

```
                private int positionTableY(int Y) {
                        //Permet de placer l'objet en fonction
de la position du curseur
                        for(i=0;i<=taille;i++) {
                            if (table_1.getRowHeight()*i <= Y && Y <
table_1.getRowHeight()*(i+1)) {
                                break;
                            }
                        }
                        return i;
                }
                private int positionTableX(int X) {
                        //Permet de placer l'objet en fonction
de la position du curseur
                        for(i=0;i<=taille;i++) {
                            if (col.getWidth()*i <= X && X <
col.getWidth()*(i+1)) {
                                break;
                            }
                        }
                return i;
                }
            });
    }
    }
```