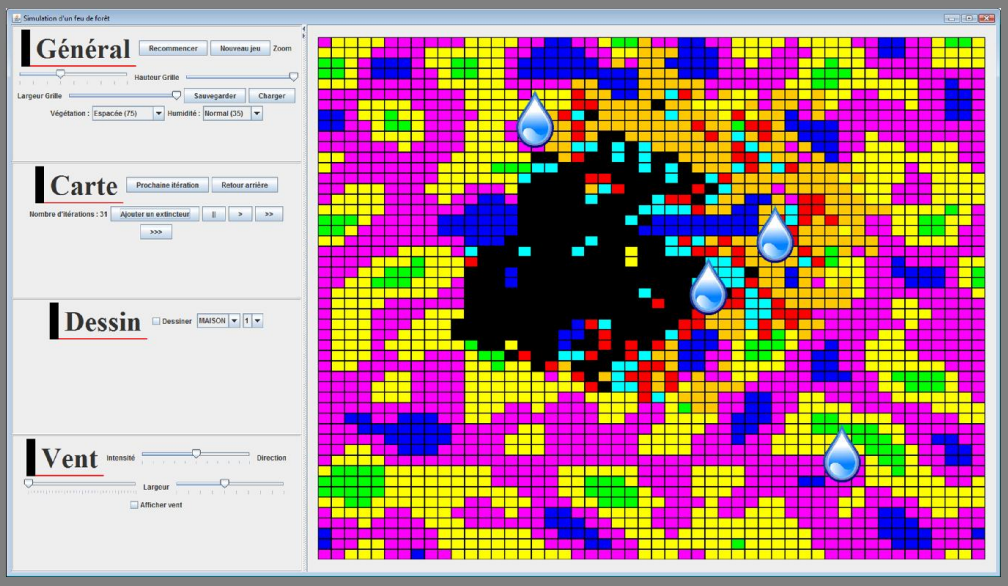
2014

Simulation d'un feu de forêt

Rapport final - juin 2014

Explications des nouveaux éléments : Heightmap, Vent et Extincteur



COURGEY Florian - GUÉNARD Thomas

02/06/2014

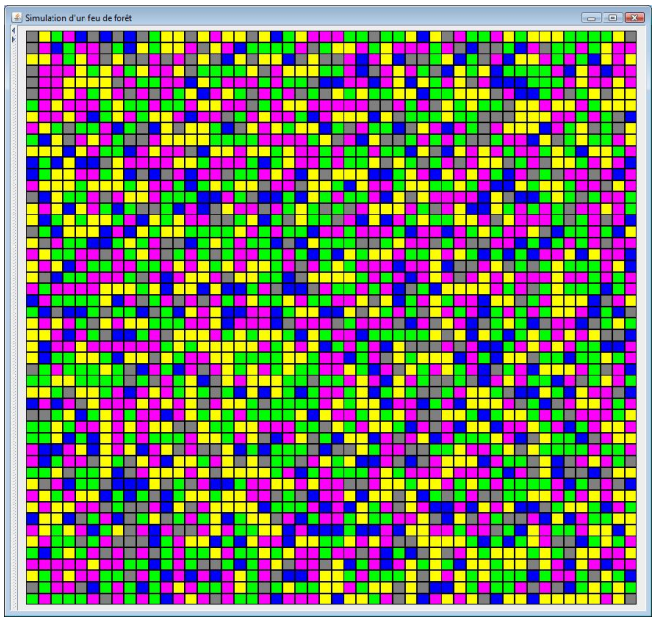
Contenu

Section 1.01 La Heightmap

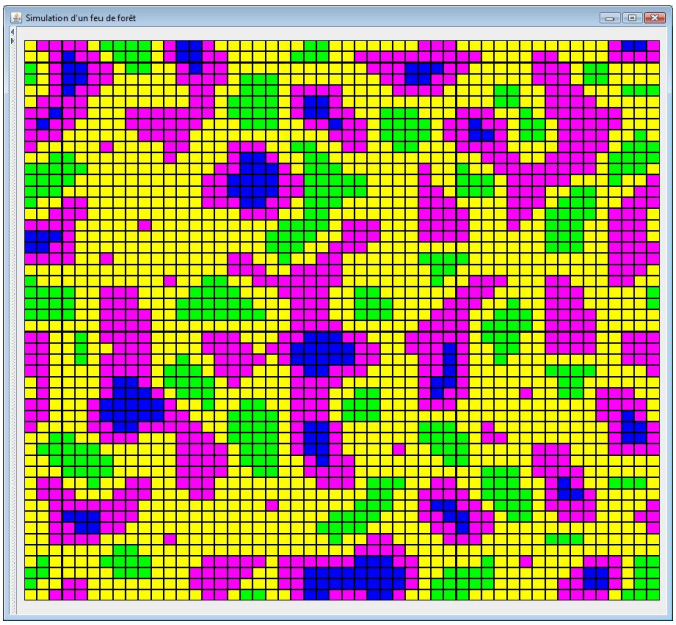
(a) Définition et exemples

La heightmap permet de générer des zones plutôt que du simple aléa :

Totalement aléatoire



Zones grâce à la heightmap

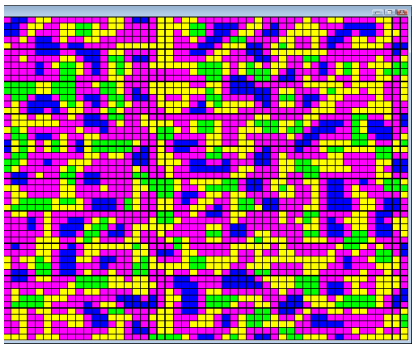


Ceci revient à créer de l'aléatoire ordonné dans un but à la fois réaliste et esthétique.

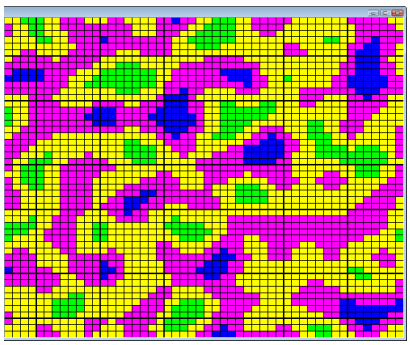
(b) Paramètre : Zoom

La valeur de zoom nous permet de créer des zones plus ou moins grandes, on zoom pour augmenter les zones, on dézoome pour avoir des zones plus petites :

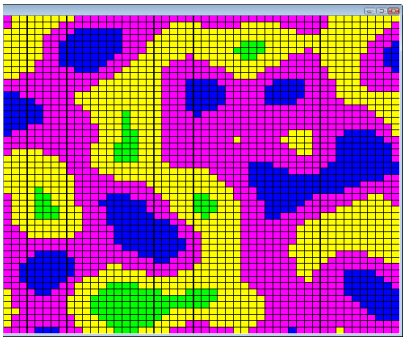
Zoom léger



Zoom moyen



Zoom fort



(c) Algorithme

Génération Heightmap

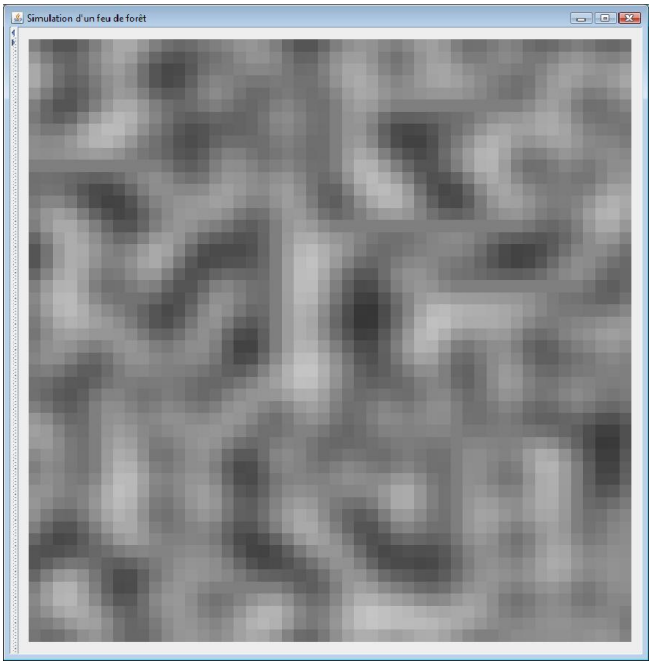
Un algorithme incompréhensible qui nous vient du site du zéro (http://fr.openclassrooms.com/informatique/cours/bruit-de-perlin) nous retourne, en lui donnant une abscisse, une ordonnée et une valeur de zoom, une valeur comprise entre 0 et 1. Comme nous voulons dans un premier temps voir si tout marche bien, nous multiplions cette valeur par 255 pour la donner en couleur d'un `JLabel`.

Dans l'exemple ci-dessous, pour x=0 et y=0, la fonction nous a retourné 127.

Pour x=2 et y=1, elle nous a retourné 103. Etc.

```
1.  127  103  85   ...  ...
2.  151  125  103  ...  ...
3.  169  142  114  ...  ...
4.  ...  ...  ...  ...  ...
```

Nous donnons ces valeurs à la couleur d'un tableau de `JLabel` pour visualiser si tout va bien :



Le résultat est concluant : Nous avons des valeurs pseudo-aléatoires qui s'échelonnent entre 50 et 200 environ.

Encore faut-il les convertir pour qu'il ne reste plus que 4 zones : Forêt, Plaine, Maison et Eau.

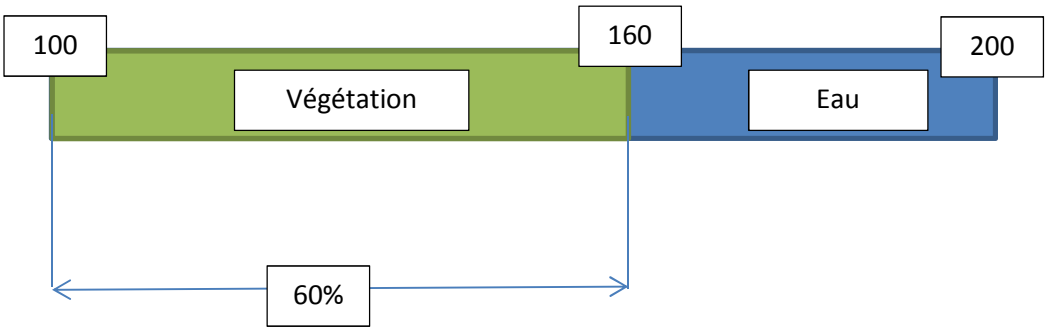
(i) Conversion en zones

Nous allons procéder par intervalles :

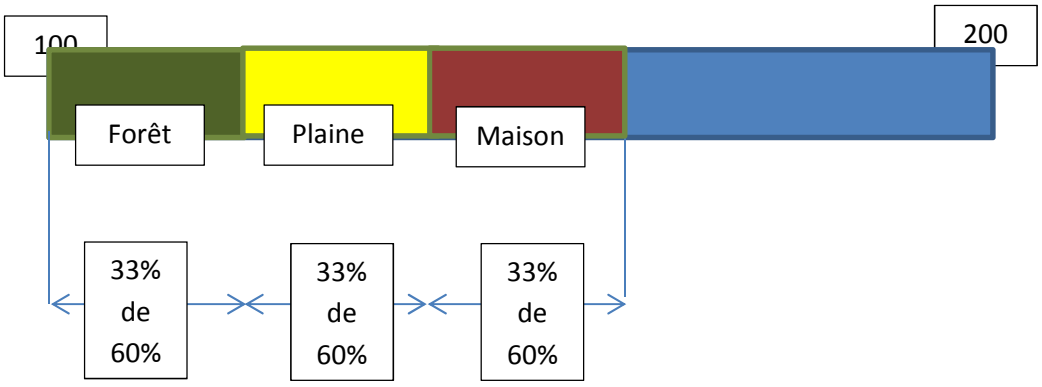
De base nous avons un premier intervalle de 100 à 200 admettons :

Nous allons le découper en deux en prenant la valeur de la végétation, prenons 60% :

Nous avons donc 2 intervalles, un premier de 100 à 160 et un second de 160 à 200 :



Nous découpons le premier en 3 parts égales afin d'avoir nos 3 types de terrain : `Forêt`, `Plaine` et `Maison` :



Ainsi, chacun des valeurs entre 100 et 120 sera une `Forêt`, entre 140 et 160 une `Maison`, etc.

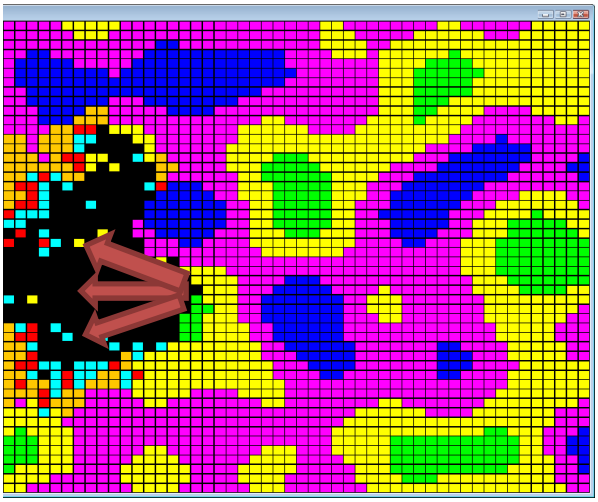
PS : On a considéré que les maisons faisaient partie de la végétation car elles sont inflammables, on peut le modifier aisément dans le code.

Section 1.02 Le Vent

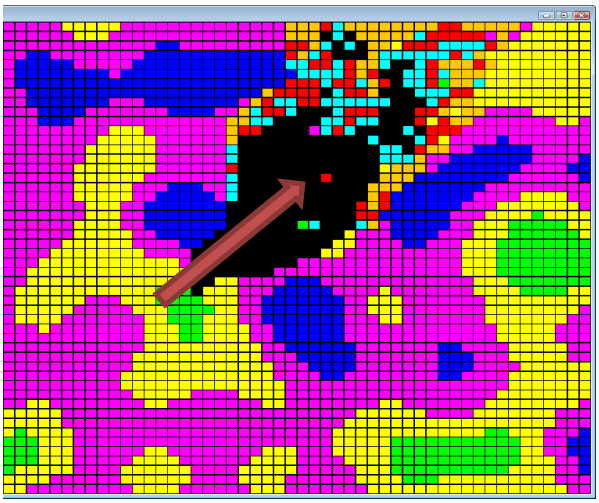
(a) Définition et exemples

Le vent permet de définir la règle de transmission des brandons d'une case aux autres.

Vent diffus, vers la gauche



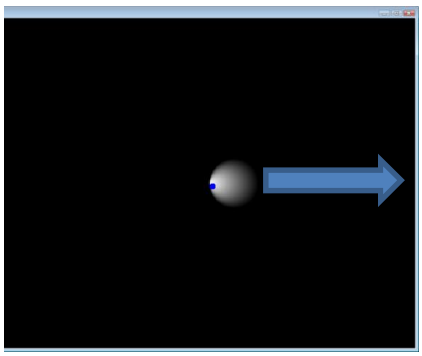
Vent fort, vers le haut-droit

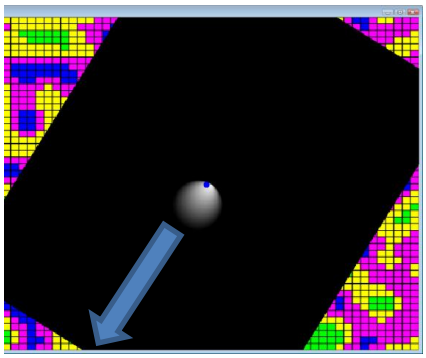


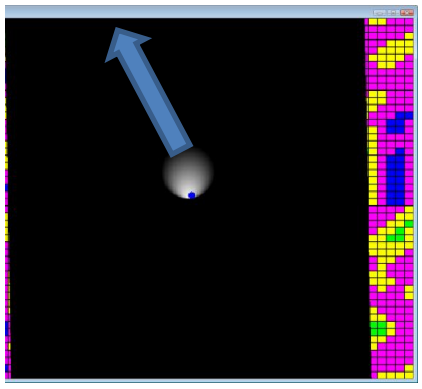
(b) Paramètre : Direction

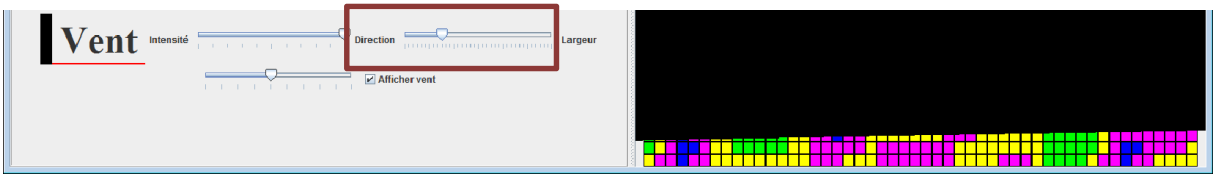
La direction est un angle en degrés, compris entre 0 et 360, qui détermine où vont aller les brandons à la prochaine itération.

Ce paramètre est réglable dans le panel des paramètres du vent :



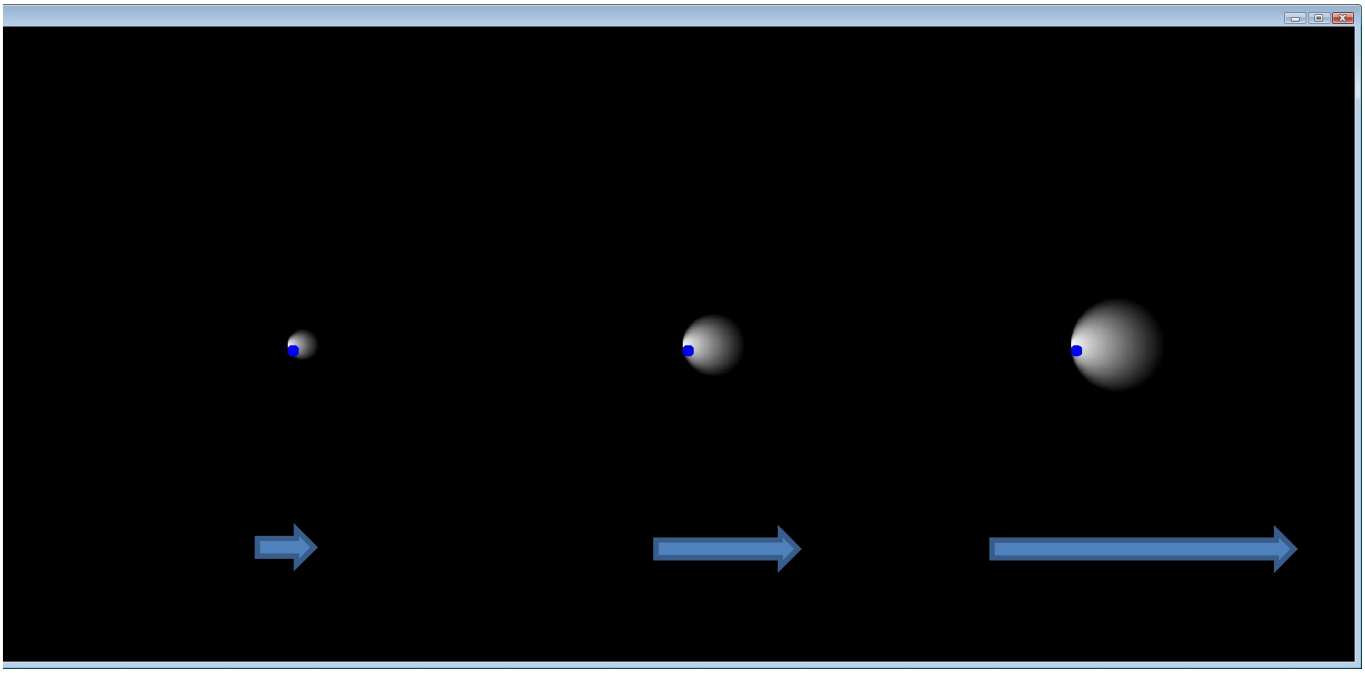






(c) Paramètre : Intensité

L'intensité détermine la force du vent : c'est la distance maximale à laquelle va être projeté un brandon :

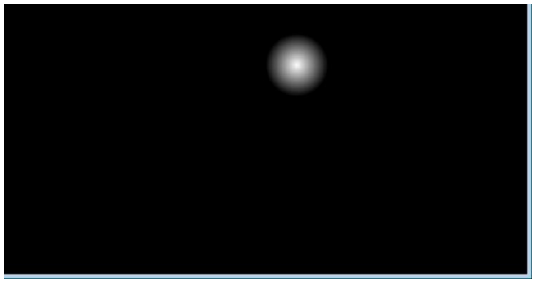


(d) Algorithme

(i) Création dégradé

Nous générons d'abord un dégradé radial, le rayon vient du `JSlider` Intensité :

```
1.  @Override
2.  protected void paintComponent(Graphics g) {
3.          super paintComponent(g);
4.
5.          float rayon = intensite*2;
6.          RadialGradientPaint rgp =
7.                  RadialGradientPaint(    centre, rayon, centreVent,
8.                                          dist, colors,
9.                                          CycleMethod.NO_CYCLE);
10.         Graphics2D g2 = (Graphics2D)g;
11.         g2.setPaint(rgp);
12. }
```



(ii) Déplacement point de focus

Le centre du dégradé est en rose.

Le point de focus est en vert.

```
1. Point2D focus = new Point2D.Float(0, (int)centre.getY());
```

(iii) Détermination du centre du vent

On a imaginé que le centre de notre vent serait à l'extrémité du dégradé radial.

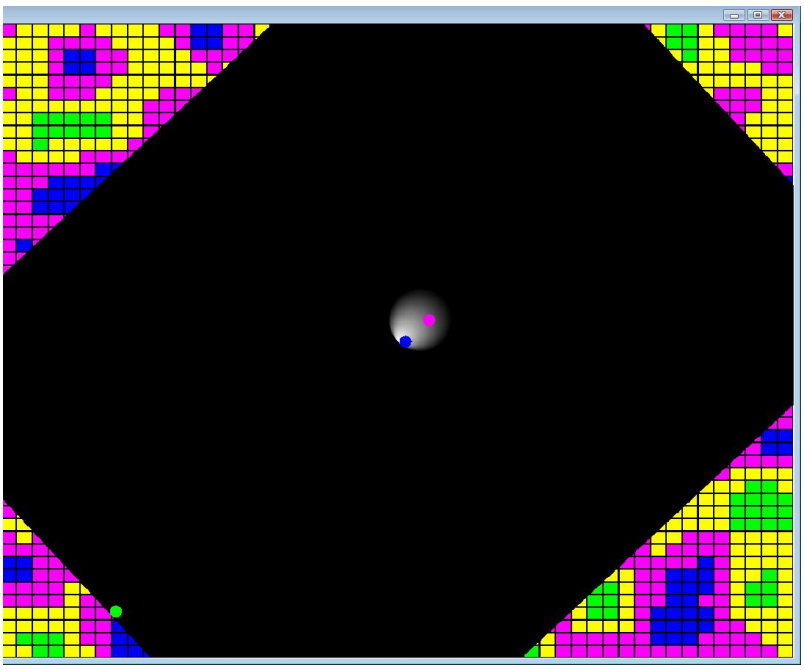
PS : Divers essais ont été menés comme placer le centre du vent à 1/3 du rayon, ou ¼, ou etc. Mais le vent se déplaçait alors plus vers l'arrière que vers l'avant.

```
1. centreVent = new Point2D.Float(      (int)centre.getX() - rayon ,
                                        (int)centre.getY());
```

(iv) Modification en échelle

L'objet `Graphics2D` permet cette option avec la méthode :

```
1. double angleRotation = Math.toRadians(-direction);
2. g2.rotate(angleRotation, centreVent.getX(), centreVent.getY());
```



(v) Conversion en matrice de probabilités

Une fois le dégradé créé, il faut le convertir en matrice de probabilités.

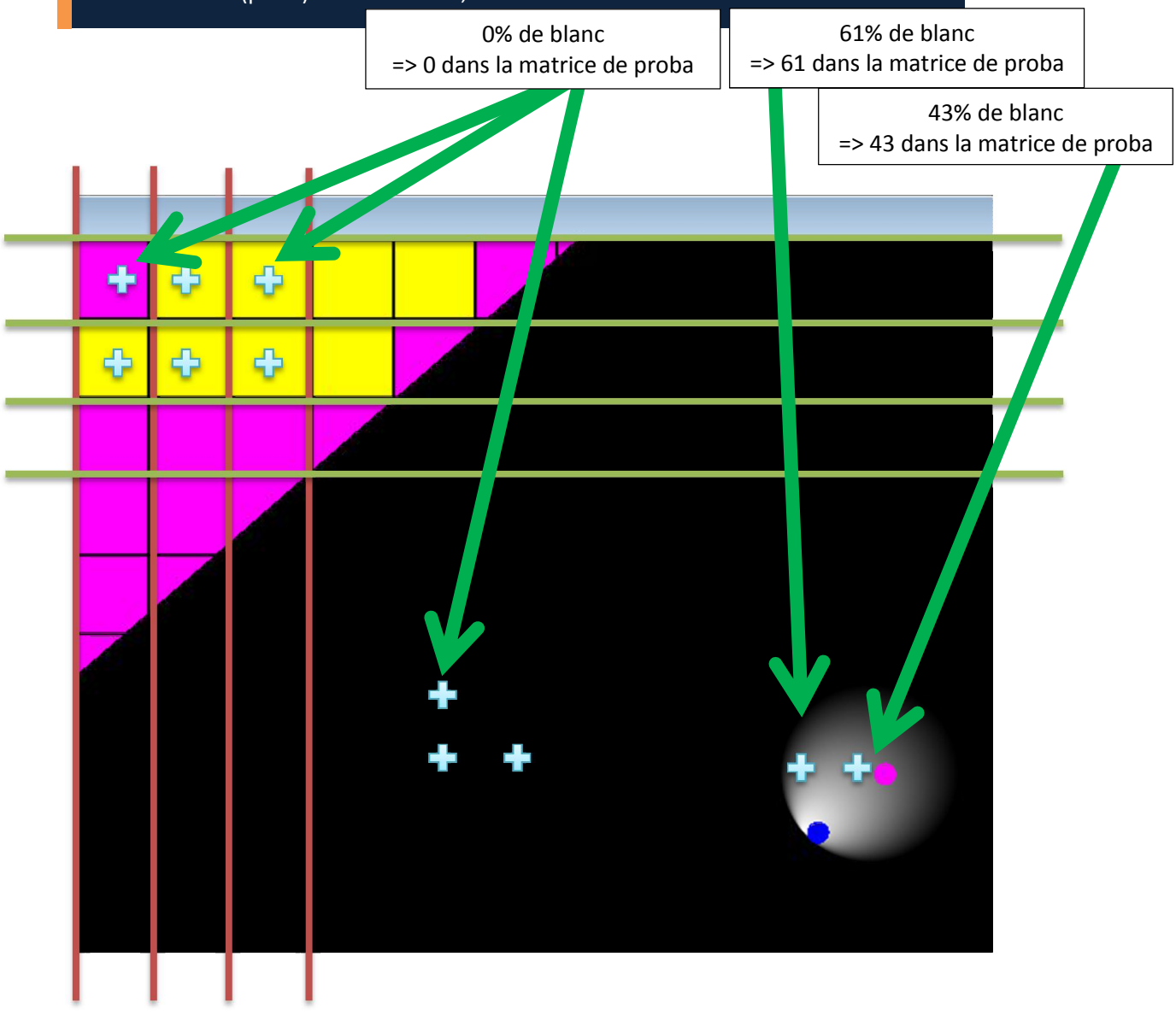
On commence par "prendre en photo le vent" en le mettant dans une `BufferedImage` afin de récupérer le pourcentage de blanc avec sa méthode getRGB().

On superpose le vent sur la grille de cases.

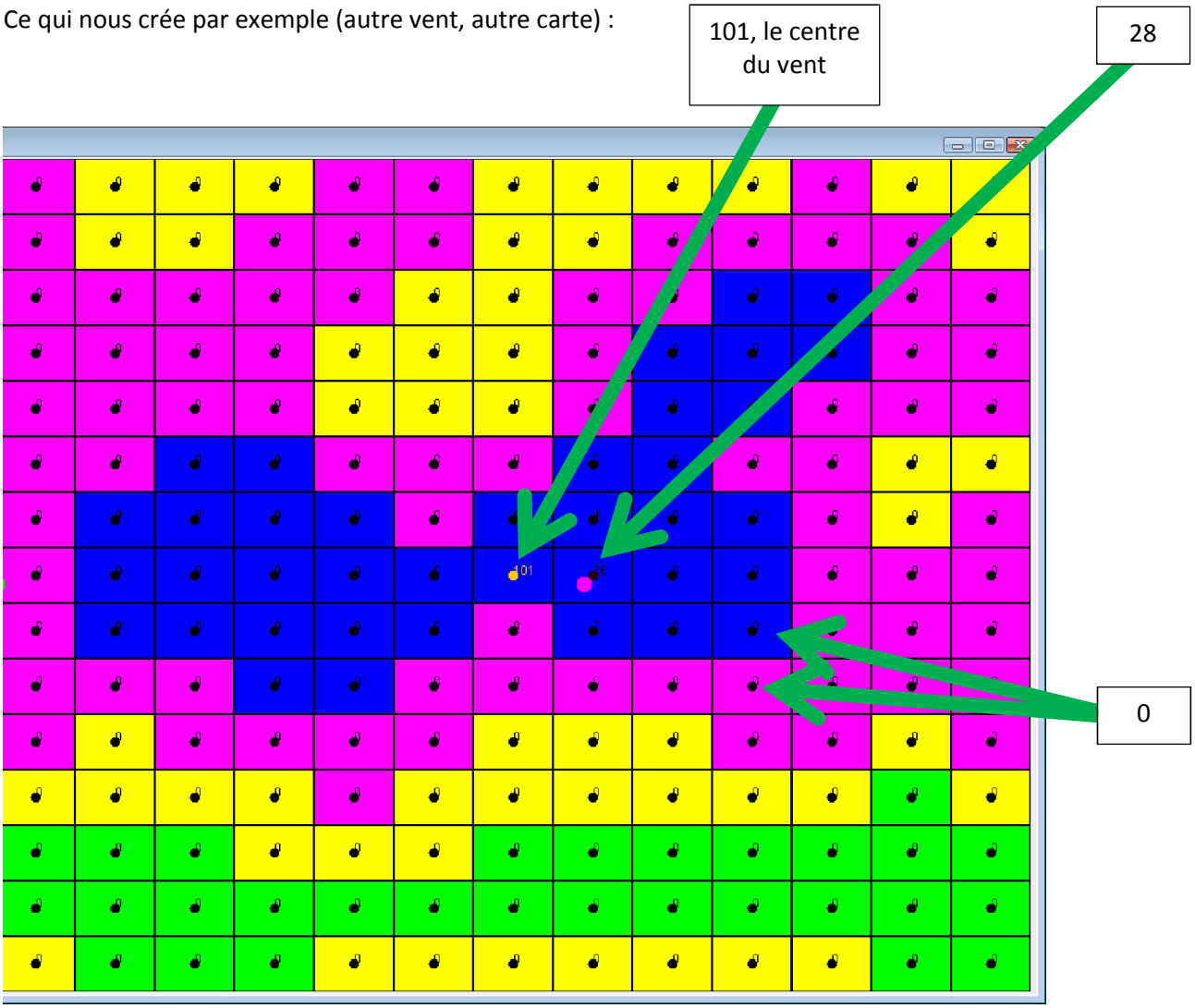
On récupère le pourcentage de blanc des centres de chacunes des cases :

(on récupère en fait le pourcentage de bleu, qui est égal au pourcentage de blanc)

```
1. int abscisse = grille[i][j].getX() + demiLargeurCase;
2. int ordonnee = grille[i][j].getY() + demiHauteurCase;
3.
4. int couleur = bi.getRGB(abscisse, ordonnee);
5. int blue = (point) & 0x000000FF;
```



Ce qui nous crée par exemple (autre vent, autre carte) :



Ce qui nous donne la matrice :

0	0	...	...	0	0
0	...	...	...	...	0
...	...	101	28	...	...
...	...	...	...	...	0
0	0	...	...	0	0

Une fois coupée avec l'outil `Matrice.crop` de Matrice.java donne :
101 28

Equivalent à :
Int [][]tab = { {101, 28} }

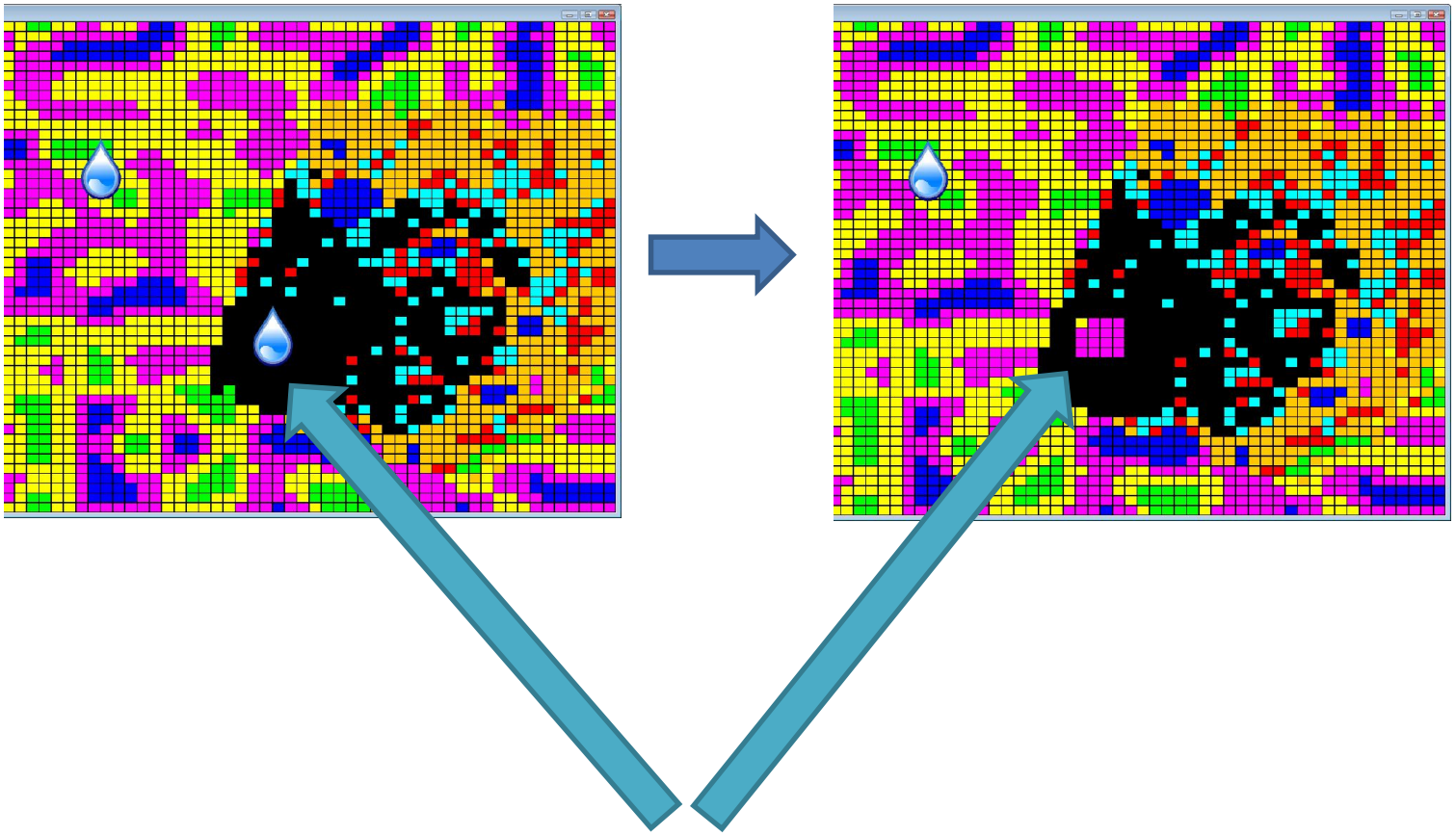
Cette matrice est ensuite superposée sur chacune des cases, comme les matrices prédéfinies que vous nous aviez proposées.

Section 1.03 L'Extincteur

(a) Définition

Un extincteur est un `JLabel` qui possède une image de fond :

Quand on clique dessus, il remet à zéro la combustion des cases qu'il couvre :



(b) Structure

(i) Structure du panel Carte

Le `JPanel` Carte est constitué de plusieurs `JPanel` superposés, du haut en bas :

- panelTransparent : reçoit les clics et contient les extincteurs

- Vent : affiche le vent quand la case Afficher Vent est cochée

- Grille : affiche la grille de cases

(ii) Structure extincteur

Case liée

```
1. // ligne de la case liée
2. private int ligne;
3. // colonne de la case liée
4. private int colonne;
```

L'extincteur est lié à une case précisément.

Quand on cliquera sur l'extincteur, il remettra à zéro toutes les cases à moins de 2 cases près de cette case liée.

Grille de cases liée

```
1. // grille de toutes les cases
2. protected Case[][] grille;
```

L'extincteur sera lié à la grille des cases pour pouvoir les éteindre.

(optimisation mineure : on pourrait définir à l'avance quelles seraient les cases à brûler et uniquement lui envoyer ces cases)

Panel lié

```
1. // panel sur lequel ajouter l'extincteur
2. private JPanel panel_transparent;
```

L'extincteur est lié à un panel pour : supprimer l'extincteur du panel quand clic et obtenir les dimensions du panel pour placer l'extincteur.

(c) Algorithme

(i) Remise à zéro de la combustion

C'est le même algorithme que celui utilisé pour "Recommencer" la simulation, dans un clickListener :

```
1.  this.addMouseListener(new MouseAdapter() {
2.          @Override
3.          public void mousePressed(MouseEvent e){
4.                  for(int i=ligne-2 ; i<=ligne+1 ; i++){
5.                          for(int j=colonne-2 ; j<=colonne+1 ; j++){
6.                                  grille[i][j].remiseAZero();
7.                          }
8.                  }
9.                  // j'enlève l'icone extincteur
10.                 panel_transparent.remove(moi);
11.                 panel_transparent.repaint();
12.         }
13. });
```

Avec remiseAZero de Case.java :

```
1.  public void remiseAZero(){
2.          nbIterationEnBruleeFroid = DEFAUT_NB_ITERATION_BRULEE_FROID;
3.          combustion = DEFAUT_COMBUSTION;
4.          nbFeu = DEFAUT_NB_FEU;
5.          setBackground(couleurBase);
6.  }
```

(ii) Ajout d'un extincteur

Dans Carte.java :

```
1.  int x = (int)(Math.random()*(largeur-2*MARGE))+MARGE;
2.  int y = (int)(Math.random()*(hauteur-2*MARGE))+MARGE;
3.
4.  Extincteur e = new Extincteur(grille[y][x], panelTransparent, grille, largeur,
                                  hauteur);
5.
6.  panelTransparent.add(e);
```

1. et 2. On choisit une abscisse et une ordonnée auxquelles sera lié l'extincteur.
4. on crée l'extincteur
6. ajout au panel