Simulation d'un feu de forêt

Voici comment on peut simuler de façon grossière mais simple un feu de forêt. Dans un quadrillage sont placés des arbres au hasard, avec une certaine densité. Ces arbres vont avoir trois états possibles : non brûlés (en vert), en feu (en rouge), consumés (en gris). Au départ, ils sont tous verts, sauf ceux qui sont sur le front de feu initial. La règle de propagation est simple : à chaque étape de temps, un arbre en feu transmet le feu à ses quatre plus proches voisins éventuels (ceux du quadrillage). Plus précisément, les arbres sont disposés à l'intérieur d'un carré. On évite de placer des arbres sur la bordure extérieure du carré, afin de contenir le feu à l'intérieur : si les points du carré ont leurs coordonnées x et y entre 0 et L, les arbres sont placés avec des x et des y entre 1 et $L - 1$. Un front de feu est disposé sur la bordure verticale gauche (en $x = 1$), avec les arbres mis en rouge. Ensuite la propagation se fait par voisinage, étape par étape. Les arbres qui étaient en feu sont considérés à l'étape suivante comme étant consumés (en gris noir). Ils ne transmettent le feu que pendant un et un seul intervalle de temps. Le phénomène s'arrête quand il n'y a plus d'arbres en flammes.

Le programme principal se réduit à la mise en place de la forêt dans le carré de côté L, à la mise à feu, puis à la propagation répétée, jusqu'à ce qu'il n'y ait plus d'arbres en feu, le nombre des arbres en feu étant placé dans la variable *nbarbresenfeu* qui évolue au fil des étapes de temps. Le test d'arrêt se produit lorsqu'il n'y a plus d'arbres en feu.

```
foret();
miseafeu();   nombreetapes=0; /* la variable nombeetapes permet de connaître le temps jusqu'à
                                 l'extinction finale */
do { propagation(); nombreetapes++; }
while(nbarbresenfeu!=0);
```

Commençons par la mise en place de la forêt. Cela revient à placer un arbre (ou pas) en chaque point du quadrillage avec une probabilité égale à la densité d choisie (par exemple $d = 0{,}6$).[1] Les points (i, j) où se trouvent les arbres sont enregistrés dans un tableau d'indices de couleurs $p[][]$ en faisant $p[i][j] = 1$ qui est l'indice du vert, tandis que les points sans arbres restent en blanc (couleur d'indice 0),[2] ce qui sous-entend que le tableau $p[][]$ est à 0 en conditions initiales.

```
void foret(void)
{ int i,j,hasard;
  srand(time(NULL)); nombrearbres=0; /* ce compteur du nombre d'arbres est facultatif */
  for(i=1; i<L; i++)  for(j=1; j<L; j++)
   { hasard=rand()%1000;
     if(hasard<(int)(d*1000.)) {p[i][j]=1;nombrearbres++;}
   }
```

/ Ce qui suit, en italiques, suppose que l'on a intégré à SDL la bibliothèque SDL_ttf qui permet d'écrire sur l'écran graphique. C'est facultatif dans un premier temps */*

[1] Pour faire cela, il suffit de tirer un nombre au hasard entre 0 et 999 (par exemple). Si le nombre obtenu est inférieur à 600 (pour une densité 0,6) on dessine un arbre au point (i, j), sinon on ne place pas d'arbre.

[2] Les couleurs sont ainsi enregistrées :

```
couleur[0]=SDL_MapRGB(screen->format,255,255,255);   /* blanc */
couleur[2]=SDL_MapRGB(screen->format,255,0,0);    /* rouge */
couleur[1]=SDL_MapRGB(screen->format,0,180,0);   /* vert */
couleur[3]=SDL_MapRGB(screen->format,100,100,100);   /* gris */
```

```
  police=TTF_OpenFont("times.ttf",20);
  sprintf( chiffre,"densite : %3.2f", (float)nombrearbres/(float)(L*L));
  texte=TTF_RenderText_Solid(police,chiffre,cblack);  position.x=600; position.y=100;
  SDL_BlitSurface(texte,NULL,screen,&position);

  for(i=1;i<L;i++) for(j=1;j<L;j++) if (p[i][j]==1)  arbre(i,j,1);
}
```

La fonction *arbre*(*x*, *y*, *icolor*) est charge de dessiner l'arbre au point *x*, *y* avec sa couleur d'indice *icolor*, verte pour le moment, sous forme d'un carré de côté *pas*. On peut prendre *pas* = 4 avec *L* = 145, ce qui revient à faire un zoom sur l'écran, un arbre étant représenté par un carré de côté 4 pixels.[3] Dans ce contexte, le carré de la forêt occupe une longueur de 145 × 4 = 580 pixels.

```
void arbre(int x,int y, int icolor)
{ int i,j;
  for(i=0;i<pas;i++) for(j=0;j<pas;j++)  putpixel(pas*x+i,pas*y+j,couleur[icolor]);
}
```

Cela étant fait, on place le front de feu en *x* = 1, ce qui revient à passer en rouge (indice 2) les arbres verts de cette ligne verticale, et l'on met dans la variable *nbarbresenfeu* le nombre d'arbres qui sont en flammes au départ.

```
void miseafeu(void)
{ int j;
  nbarbresenfeu=0;
  for(j=1;j<L;j++)
  if (p[1][j]=1) { p[1][j]=2; arbre(1,j,2); nbarbresenfeu++; }
  for(i=1;i<L;i++) for(j=1;j<L;j++) np[i][j]=p[i][j];    /* voir ci-dessous pourquoi cela */
  SDL_Flip(screen);pause();
}
```

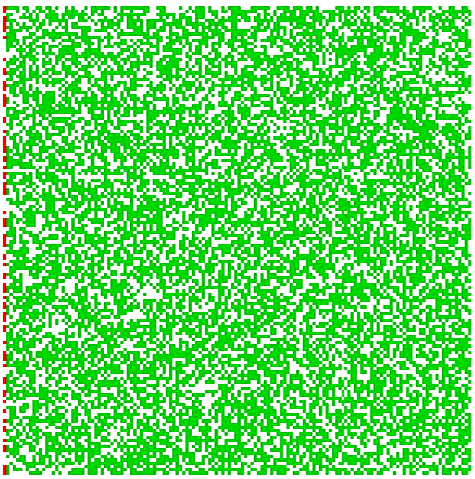
Il reste la principale fonction, celle de la propagation, qui nécessite l'usage de deux tableaux *p* et *np*.[4] A chaque étape de la propagation, ces deux tableaux sont rendus identiques au début. Remarquons que nous avons fait cela dans la fonction de mise à feu (*voir ci-dessus la ligne en italique*), avant de lancer la propagation où l'égalité des deux tableaux est faite à la fin de chaque étape de temps. Puis on parcourt le carré de la forêt en considérant les valeurs de *p*[*i*][*j*] : si l'arbre en *i*, *j* est en feu, c'est un propagateur et l'on regarde ses quatre voisins éventuels. Si l'un des voisins est un arbre vert, on le met en rouge pour l'étape suivante, en faisant cette modification dans le tableau *np*, et l'on augmente de 1 la variable *nbarbresenfeu*. Mais il faut faire attention. Il se peut qu'un voisin mis à rouge dans *np* l'ait déjà été auparavant comme voisin d'un autre arbre lors du parcours séquentiel du carré. Il convient

[3] Si l'on veut bien voir la propagation du feu, il est nécessaire de faire un tel zoom. Si l'on se contentait de prendre un arbre correspondant à un pixel d'écran, la propagation ne serait plus intéressante. Pourquoi ? Parce que l'on a pris une densité d'arbres qui est partout la même. Avec un nombre limité d'arbres (grâce au zoom), il n'y a pas d'uniformité dans la répartition. Par contre avec un grand nombre d'arbres, il y aurait tendance à l'uniformité dans la disposition des arbres, et la propagation deviendrait elle aussi régulière. Si l'on veut utiliser un arbre correspondant à un pixel, il convient de découper le carré de la forêt en plusieurs carrés plus petits, en faisant varier la densité dans chacun de ces carrés.

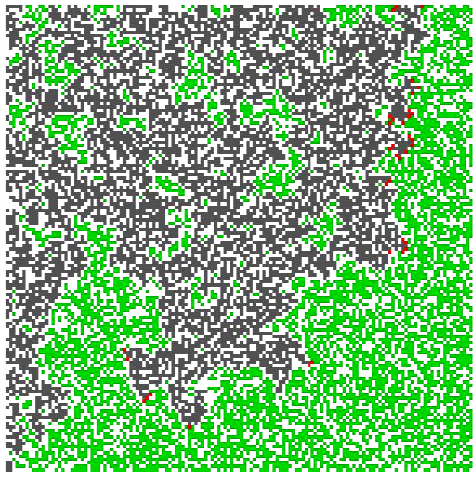
[4] Rappelons que ce genre d'algorithme de modification de paysage est synchrone (modification en parallèle) alors que l'usage d'une machine séquentielle rend indispensable le parcours séquentiel de l'écran. Ce n'est qu'après un parcours complet que l'on peut effectuer les modifications et engager l'étape suivante. D'où la nécessité de prendre deux tableaux, le deuxième enregistrant les modifications en attendant la fin du parcours. Si l'on ne prenait qu'un seul tableau, en effectuant les changements au fur et à mesure du parcours, on aboutirait à des aberrations. Par exemple, avec une rangée horizontale d'arbres où le plus à gauche serait le seul à être en feu, le parcours séquentiel de la rangée à partir du deuxième arbre mettrait le feu de proche en proche à toute la ligne d'arbres, et cela au cours d'une seule étape de temps. Ou encore en partant du premier arbre en feu, celui-ci passerait à l'état consumé, et ne mettrait le feu à aucun de ses voisins.

d'éviter de compter deux fois au lieu d'une ce passage au rouge. Enfin l'arbre propagateur en *i, j* va passer de rouge à gris, cette transformation est aussi faite dans le tableau *np*, tout en diminuant de 1 la variable *nbarbresenfeu*. Une fois le parcours terminé, le tableau *p* est mis à jour en intégrant les modifications mises dans *np*. On fait cela pour tous les arbres qui ont subi un changement (*np*[*i*][*j*] différent de *p*[*i*][*j*]). On peut alors relancer la propagation à l'étape suivante.

```
void propagation(void)
{ int i,j;
  for(i=0;i<L+1;i++)  for(j=0;j<L+1;j++)   np[i][j]=p[i][j];
  for(i=1;i<L;i++) for(j=1;j<L;j++)  if (p[i][j]==2)
   { if (p[i][j+1]==1 && np[i][j+1]!=2) {np[i][j+1]=2; nbarbresenfeu++; }
     if (p[i][j-1]==1 && np[i][j-1]!=2) {np[i][j-1]=2; nbarbresenfeu++; }
     if (p[i+1][j]==1 && np[i+1][j]!=2) {np[i+1][j]=2; nbarbresenfeu++; }
     if (p[i-1][j]==1 && np[i-1][j]!=2) {np[i-1][j]=2;  nbarbresenfeu++; }
     np[i][j]=3;nbarbresenfeu--; /* 3 est l'indice du gris, couleur de l'arbre consume */
   }
  for(i=1;i<L;i++) for(j=1;j<L;j++) if (np[i][j]!=p[i][j])
   {  p[i][j]=np[i][j];   arbre(i,j,p[i][j]);  }
  SDL_Flip(screen);SDL_Delay(20);
}
```



Forêt et mise à feu initiale



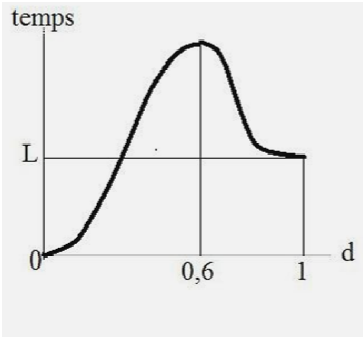
Feu en cours de propagation

Les arbres comme récepteurs, et non propagateurs

Dans le programme précédent, chaque arbre est considéré comme un émetteur éventuel : s'il brûle, il transmet le feu alentour. On peut aussi bien considérer chaque arbre comme un récepteur : s'il est vert, et qu'il a un voisin rouge, il devient rouge. La fonction *propagation* s'écrit alors plus simplement :

```
for(i=1;i<L;i++) for(j=1;j<L;j++)
if (p[i][j]==1)
    { if (p[i][j+1]==2) {np[i][j]=2; nbarbresenfeu++; }
      else if (p[i][j-1]==2) {np[i][j]=2; nbarbresenfeu++; }
      else if (p[i+1][j]==2) {np[i][j]=2; nbarbresenfeu++; }
      else if (p[i-1][j]==2) {np[i][j]=2; nbarbresenfeu++; }
     }
else if (p[i][j]==2)   {np[i][j]=3;nbarbresenfeu--;}
for(i=1;i<L;i++) for(j=1;j<L;j++) if (np[i][j]!=p[i][j])  { p[i][j]=np[i][j];  arbre(i,j,p[i][j]);}
```

Résultats



Le temps mis par le feu pour s'éteindre, en termes de nombre d'étapes, est lié à la densité de la forêt. Lorsque la densité des arbres est faible, la propagation est vite stoppée, faute de voisins à qui mettre le feu, et le temps d'extinction est petit. Lorsque la densité est forte, proche de 1, la propagation s'effectue facilement, le front de feu se déplace parallèlement à lui-même, et le temps d'extinction est proche de *L*. Entre ces deux cas extrêmes, le temps d'extinction présente un maximum très net. On assiste en effet à des phénomènes capillaires, avec des développements tentaculaires du feu dans plusieurs directions. Ce maximum est obtenu pour une densité de l'ordre de 0,6, le temps d'extinction dépassant dans ce cas 2*L*. La relation entre la densité et le temps d'extinction peut être trouvée en procédant à une série d'essais pour chaque valeur de la densité (prise de 0,1 en 0,1).

Première variante : Extinction lente

Au lieu de faire passer un arbre de l'état en feu à l'état consumé en une seule étape de temps, on va le laisser se consumer progressivement sur *P* étapes de temps (par exemple *P* = 30), en le faisant passer de rouge vif à rouge très foncé, voire noir. Cela va faire apparaître une traînée derrière les pointes rouges du feu. Les couleurs sont ainsi définies :

```
couleur[0]=SDL_MapRGB(screen->format,255,255,255);   /* blanc */
couleur[1]=SDL_MapRGB(screen->format,0,180,0);   /* vert */
couleur[2]=SDL_MapRGB(screen->format,255,0,0);    /* rouge vif */
for(i=3;i<=P+3;i++)  /* couleurs passant de rouge  à noir */
couleur[i]=SDL_MapRGB(screen->format, 220-(int)((150.*(float)(i-3)/(float)P)),0,0);
```

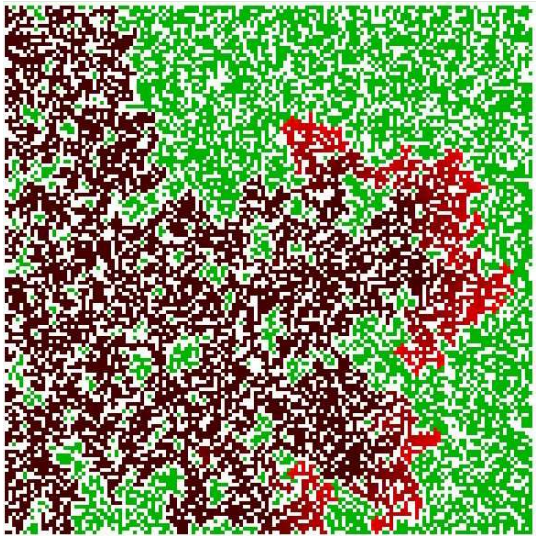
Le programme principal ne change pas par rapport à ce que l'on a fait précédemment (sauf une fonction qui s'ajoute à la fin comme on le verra), pas plus que les fonctions *foret*() et *miseafeu*(). Reste la fonction de propagation, qui doit tenir compte de l'extinction lente : lorsqu'un arbre a une couleur d'indice supérieur ou égal à 2, son indice augmente de 1.

```
void propagation(void)
{ int i,j;
  for(i=1;i<L;i++) for(j=1;j<L;j++)
  if (p[i][j]==1)
    { if (p[i][j+1]==2) {np[i][j]=2; nbarbresenfeu++; }
      else if (p[i][j-1]==2) {np[i][j]=2; nbarbresenfeu++; }
      else if (p[i+1][j]==2) {np[i][j]=2; nbarbresenfeu++; }
      else if (p[i-1][j]==2) {np[i][j]=2; nbarbresenfeu++; }
     }
  else if(p[i][j]==2) {np[i][j]=3;nbarbresenfeu--;} /* début de l'extinction */
  else if (p[i][j]>2  && p[i][j] <P+3)   {np[i][j]=p[i][j]+1;} /* évolution de l'extinction */
  for(i=1;i<L;i++) for(j=1;j<L;j++) if (np[i][j]!=p[i][j])
          {  p[i][j]=np[i][j];    arbre(i,j,p[i][j]); }
  SDL_Flip(screen);SDL_Delay(20);
}
```

La propagation s'arrête lorsqu'il n'existe plus d'arbres en feu. Mais quand cela se produit, il reste des arbres qui ne sont pas encore entièrement consumés. Il convient d'ajouter une petite fonction à la fin du programme principal qui termine le processus, soit *extinctionfinale*() ainsi programmée :

```
void extinctionfinale(void)
{  int i,j,flag;
```

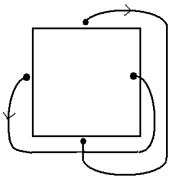
```
  do
  {flag=0;
   for(i=1;i<L;i++) for(j=1;j<L;j++)
  if (p[i][j]>2  && p[i][j]<P+3) { np[i][j]=p[i][j]+1; flag=1;} /* arbres pas encore complètement consumés */
  for(i=1;i<L;i++) for(j=1;j<L;j++) if (np[i][j]!=p[i][j])
          {  p[i][j]=np[i][j];    arbre(i,j,p[i][j]); }
  }
  while(flag==1);
}
```



Deuxième variante : Régénérescence des arbres consumés

On considère maintenant qu'après avoir fini de se consumer, comme dans le cas précédent, les arbres redeviennent verts. L'extinction graduelle du feu se fait toujours avec les couleurs dont l'indice va de 3 à $P + 3 - 1$, tandis que la dernière couleur d'indice $P + 3$ est la même que la couleur verte d'indice 1. La palette de couleurs devient :

```
couleur[0]=SDL_MapRGB(screen->format,255,255,255);   /* blanc */
couleur[1]=SDL_MapRGB(screen->format,0,200,0);   /* vert */
couleur[2]=SDL_MapRGB(screen->format,255,0,0);    /* rouge */
for(i=3;i<3+P;i++) couleur[i]=SDL_MapRGB(screen->format,220-(int)(200./((float)P)*(float)(i-3)),0,0);
couleur[3+P]=SDL_MapRGB(screen->format,0,200,0);   /* vert comme couleur[1]*/
```



Pour simplifier, et pour donner au cheminement du feu une marge de liberté plus grande, on rend la forêt cyclique. Cela signifie que ce qui déborde hors d'un côté du carré est replacé à l'intérieur, de l'autre côté. Il n'y a plus de problème de bordure, et la forêt occupe maintenant un carré de sommets opposés (0, 0) et ($L - 1$, $L - 1$). Le programme principal s'écrit :

```
  foretcyclique();
  miseafeu();
  nombreetapes=0;
  do
    { propagation();nombreetapes++;     }
  while(nombreetapes!=100);
  extinctionfinale();
```

Que va-t-il se passer ? La régénérescence régulière, et rapide (par rapport à ce qui se passe en réalité) des arbres offre de nouvelles possibilités d'expansion pour le feu. On pourrait penser que le feu

serait ainsi ranimé indéfiniment. Mais on va constater qu'il n'en est rien. Le feu va finir étouffé. Commençons par faire le programme.

- Mise en place de la forêt cyclique, la fonction *arbre*(*i*,*j*,*icolor*) restant inchangée :

```
void foretcyclique(void)
{ int i,j,hasard;
  srand(time(NULL));
  for(i=0;i<L;i++)  for(j=0;j<L;j++)     { hasard=rand()%1000;   if(hasard<(int)(d*1000.)) p[i][j]=1; }
  for(i=0;i<L;i++) for(j=0;j<L;j++) if (p[i][j]==1) arbre(i,j,1);
}
```

- Mise à feu :

Plutôt que de démarrer avec un front de feu vertical, nous allons partir d'une vingtaine d'arbres pris au hasard, auxquels est mis le feu simultanément. D'où la fonction :

```
void miseafeu(void)
{ int i,j,k;
  for(k=0;k<=20;k++)
    { do  { i=rand()%L; j=rand()%L;}  /* point (i,j) avec i et j pris au hasard entre 0 et L – 1 */
      while(p[i][j]!=1);
      p[i][j]=2; arbre(i,j,2);
    }
  for(i=0;i<L;i++) for(j=0;j<L;j++) np[i][j]=p[i][j];
  SDL_Flip(screen);
}
```

- Fonction propagation sur une étape :

La phase d'extinction d'un arbre en feu, sur *P* étapes, va être divisée en deux. Pendant la première moitié, sur *P*/2 étapes, on va considérer que l'arbre va encore propager le feu alentour, tandis que pendant la deuxième moitié, où l'arbre finit de se consumer, il ne transmet plus le feu autour de lui.

A cause de la forêt qui a été rendue cyclique, les quatre voisins du point (*i*, *j*) sont maintenant :
((*i* + 1)%*L*, *j*)
((*i* + *L* – 1)%*L*, *j*) [5]
(*i*, (*j* + 1)%*L*)
(*i*, (*j* + *L* – 1)%*L*)

Comme précédemment, à chaque étape, un arbre qui se consume voit son indice de couleur augmenter de 1, et il finit par atteindre l'indice *P* + 3. Lors de la modification finale du tableau *p*[][], via le tableau *np*[][], tous les arbres ayant la couleur *P* + 3 sont remis à la couleur 1.

```
void propagation(void)
{ int i,j;
  for(i=0;i<L;i++) for(j=0;j<L;j++)
   { if (p[i][j]==1 )
       { if (p[i][(j+1)%L]>=2 && p[i][(j+1)%L]<P/2) {np[i][j]=2;}
         else if (p[i][(j-1+L)%L]>=2  && p[i][(j-1+L)%L]<P/2) {np[i][j]=2; }
         else if (p[(i+1)%L][j]>=2 && p[(i+1)%L][j]<P/2) {np[i][j]=2;  }
         else if (p[(i-1+L)%L][j]>=2 && p[(i-1+L)%L][j]<P/2) {np[i][j]=2;}
       }
     else if (p[i][j]>=2  && p[i][j] <3+P)   {np[i][j]=p[i][j]+1;}
```

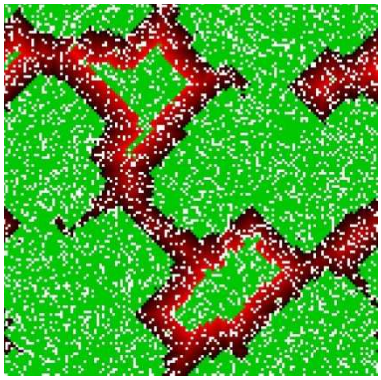
[5] Comme le modulo % en langage C ne fonctionne correctement que sur des nombres positifs ou nul, on doit prendre (*i* + *L* – 1)%*L* et non pas (*i* – 1)%*L*, car il peut arriver que *i* – 1 devienne négatif.

```
    }
  for(i=0;i<L;i++) for(j=0;j<L;j++)
  if (np[i][j]!=p[i][j])
    { p[i][j]=np[i][j];    arbre(i,j,p[i][j]);
      if (p[i][j]==3+P) p[i][j]=1;
    }
  SDL_Flip(screen);SDL_Delay(20);
}
```

- Fonction d'extinction finale :

Elle reste inchangée.

L'exécution du programme montre que le feu finit par s'éteindre, malgré le reverdissement régulier des arbres, ce qui peut sembler paradoxal. Pourquoi en est-il ainsi ?



Comme on le voit sur le dessin ci-contre[6], les fronts de feu issus des départs de feu se mêlent pour former des poches où le feu est emprisonné, avec une barrière formée d'arbres consumés qui empêchent le feu d'atteindre les arbres verts. Le feu meurt par asphyxie.

Cela peut s'appliquer à un autre problème, celui d'une épidémie. Dans ce contexte, les arbres deviennent des individus, disposés dans une zone géographique avec une densité *d*. Ils sont sains au départ (*en vert*), puis quelques individus malades (*en rouge*) sont disposés de-ci de-là. La transmission de la maladie se fait alors par voisinage. Quand un individu est touché, il reste contagieux pendant *P*/2 étapes de temps, puis pendant les *P*/2 étapes suivantes, il est immunisé et non contagieux, sur la voie de la guérison qui se produit juste après (couleur *P* + 3, il redevient vert). Le phénomène observé expérimentalement explique alors pourquoi l'épidémie finit par s'arrêter toute seule, non pas faute de « combattants », mais par sa propre asphyxie.[7]

Troisième variante : Le feu éternel

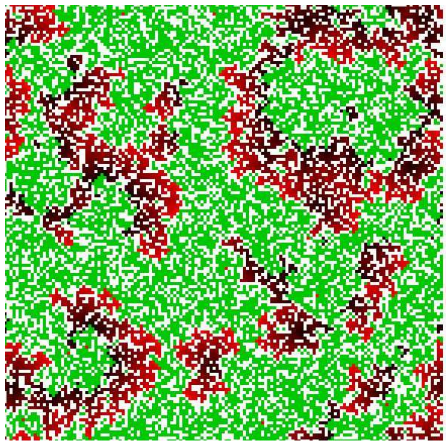
Pour empêcher l'arrêt de l'incendie, comme cela se produit dans l'exemple précédent, on doit relancer le feu régulièrement, en créant de nouveaux départs de feu localisés. Cela revient à modifier légèrement le programme principal de l'exemple précédent, toutes les autres fonctions restant identiques.

```
nombreetapes=0;
do
   { if (nombreetapes%P==0) miseafeu();  /* nouvelles mises à feu localisées toutes les P étapes de temps */
     propagation(); nombreetapes++;
   }
while(nombreetapes!=1000);
extinctionfinale();
```

On aboutit ainsi à un phénomène périodique d'expansion-régression. Un état de l'évolution est indiqué sur la figure suivante.

[6] Pour mieux voir le phénomène nous avons pris une densité forte, *d* = 0,8, ce qui provoque un front de feu d'allure rectangulaire plutôt que circulaire, cela étant dû au fait que l'on prend seulement quatre voisins autour de chaque point.

[7] La limitation de la contagion et la fin de l'épidémie n'est en général expliquée que par la mise en quarantaine des malades, ou par la miséricorde divine.



Le feu éternel