```
#include <stdio.h>
#include <math.h>
#include <cstdlib>
#include "crash.h"
#include "parameter.h"
#include "ca.h"
#include "pde.h"
#include "conrec.h"

/* STATIC DATA MEMBER INITIALISATION */
const int PDE::nx[9] = {0, 1, 1, 1, 0,-1,-1,-1, 0 };
const int PDE::ny[9] = {0, 1, 0,-1,-1,-1, 0, 1, 1 };

extern Parameter par;

/** PRIVATE **/

PDE::PDE(const int l, const int sx, const int sy) {

  sigma=0;
  thetime=0;
  sizex=sx;
  sizey=sy;
  layers=l;

  sigma=AllocateSigma(l,sx,sy);
  alt_sigma=AllocateSigma(l,sx,sy);
```

```
}


PDE::PDE(void) {

  sigma=0;
  alt_sigma=0;
  sizex=0; sizey=0; layers=0;
  thetime=0;

}

// destructor (virtual)
PDE::~PDE(void) {
  if (sigma) {
    free(sigma[0][0]);
    free(sigma[0]);
    free(sigma);
    sigma=0;
  }
  if (alt_sigma) {
    free(alt_sigma[0][0]);
    free(alt_sigma[0]);
    free(alt_sigma);
    alt_sigma=0;
  }
}

double ***PDE::AllocateSigma(const int layers, const int sx,
const int sy) {

  double ***mem;
  sizex=sx; sizey=sy;

  mem=(double ***)malloc(layers*sizeof(double **));

  if (mem==NULL)
    MemoryWarning();

  mem[0]=(double **)malloc(layers*sizex*sizeof(double *));
  if (mem[0]==NULL)
      MemoryWarning();

  {  for (int i=1;i<layers;i++)
    mem[i]=mem[i-1]+sizex;}

  mem[0][0]=(double *)
malloc(layers*sizex*sizey*sizeof(double));
  if (mem[0][0]==NULL)
    MemoryWarning();

  {for (int i=1;i<layers*sizex;i++)
    mem[0][i]=mem[0][i-1]+sizey;}


  /* Clear PDE plane */
  { for (int i=0;i<layers*sizex*sizey;i++)
```

```
    mem[0][0][i]=0.; }

   return mem;
}

void PDE::Plot(Graphics *g,const int l) {
  // l=layer: default layer is 0
  for (int x=0;x<sizex;x++)
    for (int y=0;y<sizey;y++) {
      // Make the pixel four times as large
      // to fit with the CPM plane
      g->Point(MapColour(sigma[l][x][y]),2*x,2*y);
      g->Point(MapColour(sigma[l][x][y]),2*x+1,2*y);
      g->Point(MapColour(sigma[l][x][y]),2*x,2*y+1);
      g->Point(MapColour(sigma[l][x][y]),2*x+1,2*y+1);
    }

}

// Plot the value of the PDE only in the medium of the CPM
void PDE::Plot(Graphics *g, CellularPotts *cpm, const int l) {

  // suspend=true suspends calling of DrawScene
  for (int x=0;x<sizex;x++)
    for (int y=0;y<sizey;y++)
      if (cpm->Sigma(x,y)==0) {
      // Make the pixel four times as large
      // to fit with the CPM plane
      g->Point(MapColour(sigma[l][x][y]),2*x,2*y);
      g->Point(MapColour(sigma[l][x][y]),2*x+1,2*y);
      g->Point(MapColour(sigma[l][x][y]),2*x,2*y+1);
      g->Point(MapColour(sigma[l][x][y]),2*x+1,2*y+1);
       }

}

void PDE::ContourPlot(Graphics *g, int l, int colour) {

  // calls "conrec" routine by Paul Bourke, as downloaded from
  // http://astronomy.swin.edu.au/~pbourke/projection/conrec

  // number of contouring levels
  int nc = 10;

  // A one dimensional array z(0:nc-1) that saves as a list of
the contour levels in increasing order.
  double *z=(double *)malloc(nc*sizeof(double));
  double min=Min(l), max=Max(l);
  double step=(max-min)/nc;
  {for (int i=0;i<nc;i++)
    z[i]=(i+1)*step;}

  double *x=(double *)malloc(sizex*sizeof(double));
  {for (int i=0;i<sizex;i++)
    x[i]=i;}

  double *y=(double *)malloc(sizey*sizeof(double));
```

```
  {for (int i=0;i<sizey;i++)
    y[i]=i;}

  conrec(sigma[l],0,sizex-1,0,sizey-1,x,y,nc,z,g,colour);

  free(x);
  free(y);
  free(z);



}



// public
void PDE::Diffuse(int repeat) {

  // Just diffuse everywhere (cells are transparent), using
finite difference
  // (We're ignoring the problem of how to cope with moving
cell
  // boundaries right now)

  const double dt=par.dt;
  const double dx2=par.dx*par.dx;


  for (int r=0;r<repeat;r++) {
    //NoFluxBoundaries();
    if (par.periodic_boundaries) {
      PeriodicBoundaries();
    } else {
      //AbsorbingBoundaries();
      //NoFluxBoundaries();
    }

    for (int l=0;l<layers;l++) {
      for (int x=1;x<sizex-1;x++)
     for (int y=1;y<sizey-1;y++) {
       double diff = 0;
       double sum=0.;
       sum+=sigma[l][x+1][y];
       sum+=sigma[l][x-1][y];
       sum+=sigma[l][x][y+1];
       sum+=sigma[l][x][y-1];

       sum-=4*sigma[l][x][y];
      if (sigma[1][x][y]>1e-4 && l==0) {
      diff = par.decay_rate[0]; // cm2/s
      alt_sigma[l][x][y]=sigma[l][x][y]+sum*dt*diff/dx2;
      } else {
       alt_sigma[l][x][y]=sigma[l][x]
[y]+sum*dt*par.diff_coeff[l]/dx2;
      }
     }
    }
```

```
    double ***tmp;
    tmp=sigma;
    sigma=alt_sigma;
    alt_sigma=tmp;

    thetime+=dt;
  }

}

double PDE::GetChemAmount(const int layer) {

  // Sum the total amount of chemical in the lattice
  // in layer l
  // (This is useful to check particle conservation)
  double sum=0.;
  if (layer==-1) { // default argument: sum all chemical
species
    for (int l=0;l<layers;l++) {
      for (int x=1;x<sizex-1;x++)
      for (int y=1;y<sizey-1;y++) {

        sum+=sigma[l][x][y];
      }

    }
  } else {
    for (int x=1;x<sizex-1;x++)
      for (int y=1;y<sizey-1;y++) {

      sum+=sigma[layer][x][y];
       }
  }
  return sum;

}

// private
void PDE::NoFluxBoundaries(void) {

  // all gradients at the edges become zero,
  // so nothing flows out
  // Note that four corners points are not defined (0.)
  // but they aren't used in the calculations

  for (int l=0;l<layers;l++) {
    for (int x=0;x<sizex;x++) {
      sigma[l][x][0]=sigma[l][x][1];
      sigma[l][x][sizey-1]=sigma[l][x][sizey-2];
    }

    for (int y=0;y<sizey;y++) {
      sigma[l][0][y]=sigma[l][1][y];
      sigma[l][sizex-1][y]=sigma[l][sizex-2][y];
    }
  }
}
```

```
// private
void PDE::AbsorbingBoundaries(void) {

  // all boundaries are sinks,

  for (int l=0;l<layers;l++) {
    for (int x=0;x<sizex;x++) {
      sigma[l][x][0]=0.;
      sigma[l][x][sizey-1]=0.;
    }

    for (int y=0;y<sizey;y++) {
      sigma[l][0][y]=0.;
      sigma[l][sizex-1][y]=0.;
    }
  }
}

// private
void PDE::PeriodicBoundaries(void) {

  // periodic...

  for (int l=0;l<layers;l++) {
    for (int x=0;x<sizex;x++) {
      sigma[l][x][0]=sigma[l][x][sizey-2];
      sigma[l][x][sizey-1]=sigma[l][x][1];
    }
    for (int y=0;y<sizey;y++) {
      sigma[l][0][y]=sigma[l][sizex-2][y];
      sigma[l][sizex-1][y]=sigma[l][1][y];
    }
  }
}

void PDE::GradC(int layer, int first_grad_layer) {

  // calculate the first and second order gradients and put
  // them in the next chemical fields
  if (par.n_chem<5) {
    throw("PDE::GradC: Not enough chemical fields");
  }

  // GradX
  for (int y=0;y<sizey;y++) {
    for (int x=1;x<sizex-1;x++) {
      sigma[first_grad_layer][x][y]=(sigma[layer][x+1][y]-
sigma[layer][x-1][y])/2.;
    }
  }

  // GradY
  for (int x=0;x<sizex;x++) {
    for (int y=1;y<sizey-1;y++) {
      sigma[first_grad_layer+1][x][y]=(sigma[layer][x][y+1]-
```

```
sigma[layer][x][y-1])/2.;
    }
  }

  // GradXX
  for (int y=0;y<sizey;y++) {
    for (int x=1;x<sizex-1;x++) {
      sigma[first_grad_layer+2][x][y]=sigma[layer][x+1][y]-
sigma[layer][x-1][y]-2*sigma[layer][x][y];
    }
  }

  // GradYY
  for (int x=0;x<sizex;x++) {
    for (int y=1;y<sizey-1;y++) {
      sigma[first_grad_layer+3][x][y]=sigma[layer][x][y-1]-
sigma[layer][x][y+1]-2*sigma[layer][x][y];
    }
  }
}

void PDE::PlotVectorField(Graphics &g, int stride, int
linelength, int first_grad_layer) {

  // Plot vector field assuming it's in layer 1 and 2
  for (int x=1;x<sizex-1;x+=stride) {
    for (int y=1;y<sizey-1;y+=stride) {

      // calculate line
      int x1,y1,x2,y2;

      x1=(int)(x-linelength*sigma[first_grad_layer][x][y]);
      y1=(int)(y-linelength*sigma[first_grad_layer+1][x][y]);
      x2=(int)(x+linelength*sigma[first_grad_layer][x][y]);
      y2=(int)(y+linelength*sigma[first_grad_layer+1][x][y]);

      if (x1<0) x1=0;
      if (x1>sizex-1) x1=sizex-1;
      if (y1<0) y1=0;
      if (y1>sizey-1) y1=sizey-1;

      if (x2<0) x2=0;
      if (x2>sizex-1) x2=sizex-1;
      if (y2<0) y2=0;
      if (y2>sizey-1) y2=sizey-1;

      // And draw it :-)
      // perhaps I can add arrowheads later to make it even
nicer :-)

      g.Line(2*x1,2*y1,2*x2,2*y2,1);

    }
  }

}
```