Simple Swarm

This is a partial listing of the Java code for the simple swarm program.

It includes all the code from the **Initialize** and **Evolution** sections, but does not include the variables' descriptions or the display section. The specification of the variables, for example the type of each variable and reasonable dimensions for the arrays, should be clear from the context.

Initialization section

```
/* np is the number of particles in the swarm, which can range from 2 to 96 */

np=2;
npproxy=np;

/* Select random Cartesian coordinates for the particles in a box 100 units on a side
and choose random velocities for each particle. We choose these for all particles,
including any that are not yet displayed. */

for (i=0; i<96; i++)
{
   x[i]=100.0*Math.random();
   y[i]=100.0*Math.random();
   z[i]=100.0*Math.random();

   vx[i]=(0.5-Math.random());
   vy[i]=(0.5-Math.random());
   vz[i]=(0.5-Math.random());
}

/* trace_length is the length of the tail on each particle. The length must be at least
one, since a length of 0 is interpreted as an unending trace, which will slow the
simulation. */

trace_length=12;

/* nclose is the number of neighbours of the particle with which it can intereact. */

nclose=0;

/* The degree of interaction between particles is proportional to the value of
correlation. */

correlation=1.0;
```

```
/* Create the holes that the particles should find. The positions, depths and spreads of
these holes are chosen at random. */

for (i=0; i<5; i++)
{
   blackholex[i]=100.0*Math.random();
   oldblackholex[i]=blackholex[i];
   blackholey[i]=100.0*Math.random();
   oldblackholey[i]=blackholey[i];
   blackholez[i]=100.0*Math.random();
   oldblackholez[i]=blackholez[i];
   holedepth[i]=Math.random();
   holefalloff[i]=Math.random();
}

/* Set the local best values to random values. */

for (i=0; i<97; i++)
{
   lbestx[i]=100.0*Math.random();
   lbesty[i]=100.0*Math.random();
   lbestz[i]=100.0*Math.random();
}
cycle=0;
```

Evolution section

```
/* We start with a fix to prevent the program hanging. This can occur if the slider is
used to change the value of np while the calculations in this section are being
performed. Consequently, the slider actually changes the values of  nppropxy rather
np. We reset np if needed at the start of this section. */

if (np != npproxy) np=npproxy;

/* The user may have moved one or more of the targets. If this has happened, that
information must be fed back into the simulation, otherwise the tadpoles will continue
to swarm around the spot where the target once was. If any target has been moved,
therefore, the gbest and lbest values are reset to random values. */

boolean holesmoved=false;
for (i=0; i<5; i++)
{
   if (blackholex[i]!=oldblackholex[i] || blackholey[i]!=oldblackholey[i] ||
blackholez[i]!=oldblackholez[i]) holesmoved=true;
}

if (holesmoved)
{
```

```
    for (i=0; i<5; i++)
    {
       oldblackholex[i]=blackholex[i];
       oldblackholey[i]=blackholey[i];
       oldblackholez[i]=blackholez[i];
    }
    gbest=10000.0;
    gbestx=100.0*Math.random();
    gbesty=100.0*Math.random();
    gbestz=100.0*Math.random();
    for (i=0; i<np; i++)
    {
       lbestfield[i]=10000.0;
       lbestx[i]=100.0*Math.random();
       lbesty[i]=100.0*Math.random();
       lbestz[i]=100.0*Math.random();
    }
}
cycle++;

/* Allow each particle to move. If it goes beyond the edge of the box, bring it back,
and reverse the appropriate component of velocity. */

for (i=0; i<np; i++)
{
    x[i]=x[i]+vx[i];
    if (x[i]>100.0)
    {
       x[i]=100.0;
       vx[i]=-vx[i];
    }
    if (x[i]<0.0)
    {
       x[i]=0.0;
       vx[i]=-vx[i];
    }
    y[i]=y[i]+vy[i];
    if (y[i]>100.0)
    {
       y[i]=100.0;
       vy[i]=-vy[i];
    }
    if (y[i]<0.0)
    {
       y[i]=0.0;
       vy[i]=-vy[i];
    }
    z[i]=z[i]+vz[i];
    if (z[i]>100.0)
    {
```

```
      z[i]=100.0;
      vz[i]=-vz[i];
   }
   if (z[i]<0.0)
   {
      z[i]=0.0;
      vz[i]=-vz[i];
   }
}

/* For each particle, calculate the current value of the field that it experiences */

for (i=0; i<np; i++)
{
   field=0.0;
   for (j=0; j<5; j++)
   {
      distance=Math.sqrt(Math.pow((x[i]-blackholex[j]),2)+Math.pow((y[i]-
blackholey[j]),2)+Math.pow((z[i]-blackholez[j]),2));
      field=field-1.0/(1.0+distance);
   }
   if (field<lbestfield[i])
   {
      lbestfield[i]=field;
      lbestx[i]=x[i];
      lbesty[i]=y[i];
      lbestz[i]=z[i];
      if (field<gbest)
      {
         gbest=field;
         gbestx=x[i];
         gbesty=y[i];
         gbestz=z[i];
      }
   }
}

/* Adjust the velocities so that the particles move a little towards the best local
solution */

totalv=0.0;
for (i=0; i<np; i++)
{
   vx[i]=vx[i]+0.01*(lbestx[i]-x[i])+0.001*(gbestx-x[i])+0.02*(Math.random()-0.5);
   vy[i]=vy[i]+0.01*(lbesty[i]-y[i])+0.001*(gbesty-y[i])+0.02*(Math.random()-0.5);
   vz[i]=vz[i]+0.01*(lbestz[i]-z[i])+0.001*(gbestz-z[i])+0.02*(Math.random()-0.5);
   totalv=totalv+Math.abs(vx[i])+Math.abs(vy[i])+Math.abs(vz[i]);
}

/* Finally copy the new velocities over the old ones and scale them.  */
```

```
for (i=0; i<np; i++)
{
  if(vx[i]>10.0) vx[i]=10.0;
  if(vy[i]>10.0) vy[i]=10.0;
  if(vz[i]>10.0) vz[i]=10.0;
}

double scale=2.0*np/totalv*(1.0-cycle/10000);
for (i=0; i<np; i++)
{
    vx[i]=vx[i]*scale;
    vy[i]=vy[i]*scale;
    vz[i]=vz[i]*scale;
}
```