```
 1 package vecDeffuant;
 2
 3 import java.awt.*;
 8
 9 public class Grid implements Discrete2DSpace {
10
11     /** Defines the neighbourhood to be all other agents with equal probability of being
   chosen. */
12     public static final int GLOBAL_UNIFORM = 2; // N.B. VON_NEUMANN & MOORE are 0 & 1
   respectively.
13
14     protected Object2DGrid space;
15     protected int neighbourhoodExtent;
16     protected int neighbourhoodType;
17
18     private Vector neighbourhood;
19
20     /**
21         Create the grid.
22
23         @param gridWidth, gridHeight the width and height of the grid.
24         @param torus if true, the grid wraps around.
25         @param neighbourhoodType the type of neighbourhood : VON_NEUMANN , MOORE or
   GLOBAL_UNIFORM.
26         @param neighbourhoodExtent the radius of the neighbourhood .
27         @throws IllegalArgumentException if invalid neighbourhoodType.
28     */
29     public Grid(int gridWidth, int gridHeight, boolean torus, int neighbourhoodType, int
   neighbourhoodExtent ) {
30         space = ( torus ? new Object2DTorus(gridWidth, gridHeight) : new
   Object2DGrid(gridWidth, gridHeight) );
31         if( neighbourhoodType < 0 || neighbourhoodType > 2)
32             throw new IllegalArgumentException( "Invalid neighbourhood type." );
33         this.neighbourhoodType = neighbourhoodType;
34         this.neighbourhoodExtent = neighbourhoodExtent;
35     }
36
37     /**
38         Randomly chooses a neighbour .
39
40         @param x, y the coordinates of the active agent.
41         @return a randomly chosen neighbour of the active agent.
42     */
43     public Object getNeighbour( int x, int y) {
44         switch(neighbourhoodType) {
45             case VON_NEUMANN:
46                 neighbourhood = space.getVonNeumannNeighbors( x, y, neighbourhoodExtent,
   neighbourhoodExtent, false);
47                 return(neighbourhood.get(Random.uniform.nextIntFromTo(0,
   neighbourhood.size()-1)));
48             case MOORE:
49                 neighbourhood = space.getMooreNeighbors( x, y, neighbourhoodExtent,
   neighbourhoodExtent, false);
50                 return(neighbourhood.get(Random.uniform.nextIntFromTo(0,
   neighbourhood.size()-1)));
51             case GLOBAL_UNIFORM:
52                 int jx, jy; // Coordinates of neighbouring site.
53                 // Randomly choose any site in the territory that is not the active site.
54                 do  {
55                     jx = Random.uniform.nextIntFromTo(0, space.getSizeX()-1);
56                     jy = Random.uniform.nextIntFromTo(0, space.getSizeY()-1);
57                 } while( jx == x && jy == y );
```

```
58                  return(space.getObjectAt(jx,jy));
59              }
60              return null;    // to satisfy compiler.
61          }
62
63          /**
64              Used for counting regions.
65          */
66          public Vector getVonNeumannNeighbors( int x, int y, boolean returnNulls ) {
67              return(space.getVonNeumannNeighbors( x, y, returnNulls ));
68          }
69
70          // Following methods implement Discrete2DSpace interface.
71          public int getSizeX() {return(space.getSizeX());}
72          public int getSizeY() {return(space.getSizeY());}
73          public Dimension getSize() { return(space.getSize());}
74          public Object getObjectAt( int x, int y) {return(space.getObjectAt( x,y));}
75          public double getValueAt(int x, int y) {return(space.getValueAt( x,y));}
76          public void putObjectAt(int x, int y, Object object) {space.putObjectAt( x,y,object);}
77          public void putValueAt(int x, int y, double value) {space.putValueAt( x,y,value);}
78          public BaseMatrix getMatrix() { return(space.getMatrix());}
79      }
80
```