```
 1
 2 package vecDeffuant;
 3
 4
 5 import java.awt.Color;
14
15 // This class defines the overall evolution of the system and the output data.
16
17 public class DeffuantModel extends SimModelImpl {
18
19     private ArrayList<DeffuantAgent> agentList = new ArrayList<DeffuantAgent>();
20     private DisplaySurface dsurf;   // Handles the drawing of the grid and creation of
   movies.
21     private OpenSequenceGraph graph; // A graph that plots numbers of regions and zones.
22     private int regionCount, zoneCount, disagreementCount;  // Current number of regions &
   zones and disagreements(i.e. blocked edges).
23     private float aveRegionSize, aveZoneSize; // Current average size of regions and zones.
24     private RegionCounter regionCounter;    // Object that counts regions/zones and their
   size.
25     private DataRecorder recorder;  // Use a DataRecorder object to record any relevant
   data - writes to file.
26     private Schedule schedule;
27     private Grid space; // The agents operate in a grid space defined by this object.
28     private long start; // For calculating elapsed time.
29
30     // Model parameters and their default values
31     // ---------------------------------------
32
33     /** The number of features possessed by each agent. */
34     protected int featureCount = 2;
35
36     protected int threshold =1;
37
38     protected double p=0.5;
39
40     protected double dissociating = 1;
41
42     /** Height of territory.    */
43     protected int gridHeight = 100;
44
45     /** Width of territory. */
46     protected int gridWidth = 100;
47
48     /**
49         Mutation rate for cultural drift, i.e. random mutation of agents. Mean of Poisson
   distribution from which random number
50         of mutations is generated.
51     */
52     protected double mutationProbability = 0.0;
53
54     /** Size of neighbourhood . */
55     protected int neighbourhoodExtent = 1;
56
57     /** Type of neighbourhood . The Von Neumann neighbourhood corresponds to the integer
   lattice graph */
58     protected int neighbourhoodType = Grid.VON_NEUMANN;
59
60     /** If true, the territory "wraps around" so that no agent is on an edge. */
61     protected boolean torus = true;
62
63     /** The number of traits possessed by each feature. */
64     protected int traitCount = 2;
```

```
65
66      // Simulation control parameters and their default values
67      // ------------------------------------------------------
68      /** Number of ticks between updates of display. */
69      protected int displayInterval = 1000;
70
71      /** If true, load GUI elements. */
72      protected boolean loadGui = true;
73
74      /** Number of ticks between output of data to Output window.    */
75      protected int outputInterval = 1000;
76
77      public DeffuantModel() {
78          Hashtable<Integer, String>  h1 = new Hashtable<Integer, String>();
79          h1.put(new Integer(Grid.VON_NEUMANN), "Von Neumann");
80          h1.put(new Integer(Grid.MOORE), "Moore");
81          h1.put(new Integer(Grid.GLOBAL_UNIFORM), "Global uniform");
82          ListPropertyDescriptor  pd = new ListPropertyDescriptor( "NeighbourhoodType", h1);
83          descriptors.put("NeighbourhoodType", pd);
84          addSimEventListener( new PauseListener());
85      }
86
87      // get/set methods allowing for graphical and batch manipulation of the model &
    simulation parameters.
88      //
    ------------------------------------------------------------------------------------
    -------
89      public int getDisplayInterval() {  return displayInterval; }
90      public void setDisplayInterval( int newDisplayInterval ) { displayInterval =
    newDisplayInterval; }
91      public int getFeatureCount() {  return featureCount; }
92      public void setFeatureCount( int newFeatureCount ) {featureCount =
    Math.max(1,newFeatureCount );}
93      public int getGridWidth() {  return gridWidth; }
94      public void setGridWidth( int newGridWidth ) { gridWidth = newGridWidth; }
95      public int getGridHeight() {  return gridHeight; }
96      public void setGridHeight( int newGridHeight ) { gridHeight = newGridHeight; }
97      public boolean isLoadGui() {return loadGui;}
98      public void setLoadGui( boolean b) {this.loadGui = b;}
99      public double getMutationProbability() {  return mutationProbability; }
100     public void setMutationProbability( double newMutationProbability ) {
101         mutationProbability = newMutationProbability;
102         if(mutationProbability < 0) mutationProbability = 0;
103         else if(mutationProbability > 1) mutationProbability = 1;
104     }
105     public int getNeighbourhoodExtent() { return neighbourhoodExtent;}
106     public void setNeighbourhoodExtent( int neighbourhoodExtent ) {this.neighbourhoodExtent =
    neighbourhoodExtent;}
107     public int getNeighbourhoodType() { return neighbourhoodType;}
108     public void setNeighbourhoodType( int neighbourhoodType ) {this.neighbourhoodType =
    neighbourhoodType;}
109     public int getOutputInterval() {  return outputInterval; }
110     public void setOutputInterval( int newOutputInterval ) { outputInterval =
    newOutputInterval; }
111     public int getThreshold() {  return threshold; }
112     public void setThreshold( int newThreshold ) { threshold = newThreshold; }
113     public double getP() { return p; }
114     public void setP(double newP) {
115         p = newP;
116         if(p < 0) p = 0;
117         else if(p > 1) p = 1;
118     }
```

```
119     public boolean isTorus() { return torus; }
120     public void setTorus(boolean b) { torus = b; }
121     public double getDissociating() { return dissociating; }
122     public void setDissociating( double newDissociating ){
123         dissociating = newDissociating;
124         if(dissociating < 0) dissociating = 1;
125         else if(dissociating > 2) dissociating = 2;
126     }
127 
128     /**
129         Begins a simulation run. All initialization, building the model, display, etc.
    takes
130         place here. This method is called whenever the Start button (or the Step button if
    the run
131         has not yet begun) is clicked.
132         If running in batch mode this is called to kick off a new simulation run.
133     */
134     public void begin() {
135         buildModel();
136         if (loadGui) {
137             buildDisplay();
138             graph.display();
139         }
140         buildSchedule();
141         if (dsurf != null && loadGui)
142             dsurf.display();
143         start = System.currentTimeMillis(); // for timing.
144     }
145 
146     // Builds the display
147     private void buildDisplay() {
148         Object2DDisplay agentDisplay = new Object2DDisplay( space);
149         agentDisplay.setObjectList( agentList );
150 
151         dsurf.addDisplayableProbeable( agentDisplay, "Agents" );
152         dsurf.setBackground(java.awt.Color. white);
153         addSimEventListener( dsurf);
154 
155         // Set up graph.
156         graph.addSequence( "Regions", new Sequence() {
157             public double getSValue() { return regionCount; }},
158             Color. red, OpenSequenceGraph. FILLED_CIRCLE );
159         graph.addSequence( "Zones", new Sequence() {
160             public double getSValue() { return zoneCount; }},
161             Color. blue, OpenSequenceGraph. CIRCLE);
162         graph.setAxisTitles( "Time", "Counts" );
163         graph.setXRange(0, 50000);
164         graph.setYRange(0, agentList.size());
165         graph.setSize(600, 400);
166     }
167 
168     /**
169         Builds the model. Called from begin().
170     */
171     protected void buildModel() {
172         int i;
173 
174         // Get the displayed random seed value from the RePast Parameters panel and use it
    to initialize the random number generator.
175         BaseController controller = (BaseController) this.getController();
176         long seed = controller.getRandomSeed();
177         this.setRngSeed( seed);  // same effect as Random.setSeed(seed).
```

```
178
179         Random.createUniform(); // Creates Colt object: static cern.jet.random.Uniform
    uniform.
180         space = new Grid(gridWidth, gridHeight, torus, neighbourhoodType,
    neighbourhoodExtent);
181         AgentColour siteColour = new AgentColour(featureCount, traitCount); // Create
    object for colouring the agents.
182
183         int[] randomTraits = new int[featureCount];
184         for (int x = 0; x < gridWidth; x++) {
185             for (int y = 0; y < gridHeight; y++) {
186
187                 // Create random trait values for the agent.
188                 for (i = 0; i < featureCount; i++) {
189                     //randomTraits[i] =0;
190                     randomTraits[i] = Random.uniform.nextIntFromTo(0, traitCount-1);     //
    Colt method.
191                 }
192
193                 // Create the agent and add it to the space.
194                 DeffuantAgent agent = new DeffuantAgent(x, y, space, featureCount,
    traitCount, randomTraits, /*negate,*/ siteColour, threshold, p, dissociating);
195                 agentList.add(agent);
196                 space.putObjectAt(x, y, agent);
197             }
198         }
199
200         regionCounter = new RegionCounter(featureCount, agentList, space, threshold,
    dissociating);  // Create the object that counts the regions and zones.
201
202         initDataRecorder();
203     }
204
205     // Builds the simulation schedule. Called from begin().
206     private void buildSchedule() {
207         schedule.scheduleActionBeginning(0, new Interaction() );     // Core interaction of
    the vec. Deffuant model is executed at every tick.
208
209         if( mutationProbability > 0.0 )
210             schedule.scheduleActionBeginning(0, new Mutation() );
211
212         CountAction countAction = new CountAction();
213         OutputAction outputAction = new OutputAction();
214         StopAction stopAction = new StopAction();
215
216         // Create ActionGroup to ensure that regions/zones are counted before outputted  .
217         ActionGroup actionGroup = new ActionGroup();
218         actionGroup.addAction(countAction);
219         if(loadGui) {
220             actionGroup.addAction(outputAction);
221             actionGroup.addAction(new BasicAction() {
222                 public void execute() {graph.step();}} );
223         }
224         actionGroup.addAction(stopAction);
225         schedule.scheduleActionAt( 1, actionGroup, 1 );
226         schedule.scheduleActionAtInterval( outputInterval, actionGroup );
227         schedule.scheduleActionAtEnd(recorder, "record");
228         schedule.scheduleActionAtEnd(recorder, "writeToFile");
229     }
230
231     public String[] getInitParam() {
232         // Note order of strings determines non-alpha order of parameters in Repast model
```

```
    settings window.
233         String[] params = { "gridWidth", "gridHeight", "torus", "neighbourhoodType",
    "neighbourhoodExtent",
234             "featureCount","threshold", "traitCount", "mutationProbability","p",
    "dissociating",
235             "displayInterval", "outputInterval", "loadGui" };
236         return params;
237     }
238 
239     public String getName() { return "vec. Deffuant model"; }
240 
241     public int getRegionCount() { return regionCount; }
242 
243     public Schedule getSchedule() { return schedule; }
244 
245     /**
246         Returns all configurations of all agents in a single string.
247     */
248     public String getTraits() {
249         int n = agentList.size();
250         DeffuantAgent agent;
251         StringBuffer sb = new StringBuffer(13+n*(1+featureCount*2));    // Intitial Size??
252         sb.append("\nTrait values");
253         for(int i = 0; i < n; i++ ) {
254             agent = (DeffuantAgent) agentList.get(i);
255             sb.append("\n"+agent.traitsToString());
256         }
257         return sb.toString();
258     }
259 
260     public int getZoneCount() { return zoneCount; }
261 
262     public float getZoneSize() { return aveZoneSize; }
263 
264     public float getRegionSize() { return aveRegionSize; }
265 
266     public int getDisagreementCount() { return disagreementCount;}
267 
268     /** Writes simulated data to file.  */
269     protected void initDataRecorder() {
270         String header = "vec. Deffuant model\nRandom seed: " +getRngSeed();
271         recorder = new DataRecorder("./models/vec. Deffuant.txt", this, header );
272         recorder.createNumericDataSource(" ", this, "getRegionCount", -1, -1);
273         recorder.createNumericDataSource(" ", this, "getZoneCount", -1, -1);
274         recorder.createNumericDataSource(" ", this, "getRegionSize", -1, -1);
275         recorder.createNumericDataSource(" ", this, "getZoneSize", -1, -1);
276         recorder.createNumericDataSource(" ", this, "getDisagreementCount", -1, -1);
277     }
278 
279     /**
280         Called whenever the Setup Model button (2 arrows) is clicked.
281         Also called when the model is first loaded.
282     */
283     public void setup() {
284         if (dsurf != null) {dsurf.dispose();}
285 
286         dsurf = null;
287         schedule = null;
288         if (graph != null) graph.dispose();
289         graph = null;
290 
291         System.gc();
```

```
292
293         String displayTitle = "vec. Deffuant Model Display" ;
294         String plotTitle = "vec. Deffuant Time Series Plot" ;
295         if (loadGui) {
296             dsurf = new DisplaySurface( this, displayTitle );
297             this.registerDisplaySurface( displayTitle, dsurf);
298             graph = new OpenSequenceGraph( plotTitle, this);
299             this.registerMediaProducer( plotTitle, graph); // ??
300         }
301         schedule = new Schedule(1);
302         agentList = new ArrayList<DeffuantAgent>();
303         space = null;
304         recorder = null;
305     }
306
307     public static void main(String[] args) {
308         SimInit init = new SimInit();
309         DeffuantModel model = new DeffuantModel();
310         init.loadModel( model, null, false);
311     }
312
313     // Counts the number of regions and zones.
314     class CountAction extends BasicAction {
315         public void execute() { // Count the number of regions and zones.
316             regionCount = regionCounter.countRegions();
317             zoneCount = regionCounter.countZones();
318             aveZoneSize = regionCounter.aveZonesSize();
319             aveRegionSize = regionCounter.aveRegionSize();
320             disagreementCount = regionCounter.countDisagreements();
321             }
322         }
323
324
325     /*
326         This implements the core interaction of the vec . Deffuant model.
327         An agent is picked at random and its step() method is called.
328         The step() method picks a neighbouring agent at random and initiates interaction
329         according to the vec . Deffuant Model.
330         A total of 10000 interaction is attempted until the configuration of a site
331         changes (successful interaction).
332         This action is executed at every tick of the simulation.
333     */
334     class Interaction extends BasicAction {
335         public void execute() {
336         boolean event= false;
337         int bitCount=1;
338             do{
339             int i = Random.uniform.nextIntFromTo(0, agentList.size()-1);     // Colt method.
340             DeffuantAgent agent = (DeffuantAgent) agentList.get(i);
341
342             if(++bitCount < 10000) event = agent.step(agent);
343             else event = true;
344
345             }while(event = false);
346
347             // Update grid screen display as necessary.
348             if(dsurf != null && loadGui ) {
349                 if( displayInterval > 1 ) {
350                     if((DeffuantModel.this.getTickCount() % displayInterval ) == 0)
    dsurf.updateDisplay();
351                 }
352                 else if(event) dsurf.updateDisplay();
```

```
353             }
354         }
355     }
356 
357     /*
358         This implements random mutation (cultural drift) in the vec  . Deffuant model.
359     */
360     class Mutation extends BasicAction {
361         public void execute() {
362             if( Random.uniform.nextDoubleFromTo(0, 1) <=  mutationProbability  ) {
363                 // Pick an agent at random and mutate it.
364                 DeffuantAgent agent = (DeffuantAgent)
    agentList.get(Random.uniform.nextIntFromTo(0, agentList.size()-1));
365                 agent.mutate();
366             }
367         }
368     }
369 
370     // Outputs the number of ticks, regions and zones to the Output window.
371     class OutputAction extends BasicAction {
372         public void execute() {
373             System.out.println((long)getTickCount()+" ticks: "+regionCount+" regions,
    "+zoneCount+" zones, "+aveZoneSize+" aveZoneSize, "+aveRegionSize+" aveRegionSize,
    "+disagreementCount+" disagreements " );
374         }
375     }
376 
377     // Updates the agent display when user clicks on pause or stop button.
378     class PauseListener implements SimEventListener {
379         public void simEventPerformed(SimEvent  evt) {
380             if( evt.getId() == SimEvent.PAUSE_EVENT || evt.getId() == SimEvent.STOP_EVENT)
381                 if (dsurf != null && loadGui)
382                     dsurf.updateDisplay();
383         }
384     }
385 
386     /*
387         Stops the simulation when the model converges to stable regions (zones).
388         The final number of regions/zones is output.
389         The elapsed time is calculated and output.
390     */
391     class StopAction extends BasicAction {
392         public void execute() {
393             if(regionCount < zoneCount) {
394                 System.out.println((long)getTickCount()+" ticks: "+regionCount+" regions,
    "+zoneCount+" zones " );
395                 long stop = System.currentTimeMillis();
396                 System.out.println("Converged: elapsed time = " +(stop-start)/1000+" secs");
397                 stop();
398                 }
399         }
400     }
401 }
402 
```