```java
package vecDeffuant;

import java.util.*;

// This class does the same as DeffuantModel without loading the GUI graphical simulation and terminates the simulation after 100000000 ticks. Used for Monte  Carlo simulations.
public class DeffuantBatchModel  extends SimModelImpl {

    private ArrayList<DeffuantAgent>  agentList = new ArrayList<DeffuantAgent>();
    private OpenSequenceGraph  graph;
    private int regionCount, zoneCount, disagreementCount;
    private float aveRegionSize, aveZoneSize;
    private RegionCounter  regionCounter;
    private DataRecorder  recorder;
    private Schedule  schedule;
    private Grid space;
    private long start;

    /** The number of features possessed by each agent. */
    protected int featureCount = 2;

    protected int threshold =1;

    protected double p=0.5;

    protected double dissociating = 1;

    /** Height of territory.    */
    protected int gridHeight = 100;

    /** Width of territory. */
    protected int gridWidth = 100;

    /**
        Mutation rate for cultural drift. Mean of Poisson  distribution from which random number
        of mutations is generated.
    */
    protected double mutationProbability  = 0.0;

    /** Size of neighbourhood . */
    protected int neighbourhoodExtent  = 1;

    /** Type of neighbourhood . */
    protected int neighbourhoodType  = Grid.VON_NEUMANN;

    /** If true, the territory "wraps around" so that no agent is on an edge. */
    protected boolean torus = true;

    /** The number of traits possessed by each feature. */
    protected int traitCount = 2;


    /** Number of ticks between output of data to Output window.    */
    protected int outputInterval  = 1000;

    public int getFeatureCount() {  return featureCount; }
    public void setFeatureCount( int newFeatureCount ) {featureCount = Math.max(1,newFeatureCount );}
    public int getGridWidth() {  return gridWidth; }
    public void setGridWidth( int newGridWidth ) { gridWidth = newGridWidth; }
    public int getGridHeight() {  return gridHeight; }
```

```
66      public void setGridHeight( int newGridHeight ) { gridHeight = newGridHeight; }
67      public double getMutationProbability() {  return mutationProbability ; }
68      public void setMutationProbability( double newMutationProbability ) {
69          mutationProbability = newMutationProbability ;
70          if(mutationProbability < 0) mutationProbability = 0;
71          else if(mutationProbability > 1) mutationProbability = 1;
72      }
73      public int getNeighbourhoodExtent() { return neighbourhoodExtent ;}
74      public void setNeighbourhoodExtent( int neighbourhoodExtent ) {this.neighbourhoodExtent =
    neighbourhoodExtent ;}
75      public int getNeighbourhoodType() { return neighbourhoodType ;}
76      public void setNeighbourhoodType( int neighbourhoodType ) {this.neighbourhoodType =
    neighbourhoodType ;}
77      public int getOutputInterval() {  return outputInterval ; }
78      public void setOutputInterval( int newOutputInterval ) { outputInterval =
    newOutputInterval ; }
79      public int getTraitCount() {  return traitCount; }
80      public void setTraitCount( int newTraitCount ) { traitCount = newTraitCount; }
81      public int getThreshold() {  return threshold; }
82      public void setThreshold( int newThreshold ) { threshold = newThreshold; }
83      public double isDissociating() {  return dissociating; }
84      public double getDissociating() {  return dissociating; }
85      public void setDissociating( double newDissociating ){
86          dissociating = newDissociating;
87          if(dissociating < 0) dissociating = 1;
88          else if(dissociating > 2) dissociating = 2;
89      }
90      public double getP() {  return p; }
91      public void setP(double newP) {
92          p = newP;
93          if(p < 0) p = 0;
94          else if(p > 1) p = 1;
95      }
96      public boolean isTorus() {  return torus; }
97      public void setTorus(boolean b) { torus = b; }
98
99
100     public void begin() {
101         buildModel();
102         buildSchedule();
103         start = System.currentTimeMillis(); // for timing.
104     }
105
106
107     protected void buildModel() {
108         int i;
109
110         BaseController controller = (BaseController) this.getController();
111         long seed = controller.getRandomSeed();
112         this.setRngSeed( seed);
113
114         Random.createUniform();
115         space = new Grid(gridWidth, gridHeight, torus, neighbourhoodType,
    neighbourhoodExtent );
116         AgentColour siteColour = new AgentColour( featureCount, traitCount);
117
118         int[] randomTraits = new int[featureCount];
119         for (int x = 0; x < gridWidth; x++) {
120             for (int y = 0; y < gridHeight; y++) {
121
122                 for (i = 0; i < featureCount; i++) {
123                     randomTraits[i] = Random.uniform.nextIntFromTo(0, traitCount-1);
```

```
124                 }
125
126                 DeffuantAgent agent = new DeffuantAgent( x, y, space, featureCount,
    traitCount, randomTraits, /*negate,*/ siteColour, threshold, p, dissociating);
127                 agentList.add(agent);
128                 space.putObjectAt( x, y, agent);
129             }
130         }
131
132         regionCounter = new RegionCounter( featureCount, agentList, space, threshold,
    dissociating);
133         initDataRecorder();
134     }
135
136     private void buildSchedule() {
137         schedule.scheduleActionBeginning(0,  new Interaction() );
138
139         if( mutationProbability > 0.0 )
140             schedule.scheduleActionBeginning(0,  new Mutation() );
141
142         CountAction countAction = new CountAction();
143         OutputAction outputAction = new OutputAction();
144         StopAction stopAction = new StopAction();
145
146         ActionGroup actionGroup = new ActionGroup();
147         actionGroup.addAction( countAction );
148         actionGroup.addAction( outputAction );
149         actionGroup.addAction( stopAction );
150         schedule.scheduleActionAt( 1,  actionGroup, 1 );
151         schedule.scheduleActionAtInterval(  outputInterval, actionGroup );
152         schedule.scheduleActionAtEnd( recorder, "record");
153         schedule.scheduleActionAtEnd( recorder, "writeToFile");
154     }
155
156     public String[] getInitParam() {
157
158         String[] params = { "gridWidth", "gridHeight", "torus", "neighbourhoodType",
    "neighbourhoodExtent",
159             "featureCount","threshold", "traitCount", "mutationProbability","p",
    "dissociating",
160             "displayInterval", "outputInterval", "loadGui" };
161         return params;
162     }
163
164     public String getName() {  return "vec. Deffuant Batch model" ; }
165
166     public int getRegionCount() {  return regionCount; }
167
168     public Schedule getSchedule() {  return schedule; }
169
170
171     public String getTraits() {
172         int n = agentList.size();
173         DeffuantAgent agent;
174         StringBuffer sb = new StringBuffer(13+n*(1+featureCount*2));
175         sb.append("\nTrait values");
176         for(int i = 0; i < n; i++ ) {
177             agent = (DeffuantAgent) agentList.get(i);
178             sb.append("\n"+agent.traitsToString());
179         }
180         return sb.toString();
181     }
```

```
182
183     public int getZoneCount() {  return zoneCount; }
184
185     public float getZoneSize() {  return aveZoneSize; }
186
187     public float getRegionSize() {  return aveRegionSize; }
188
189     public int getDisagreementCount() { return disagreementCount ;}
190
191     /** Writes simulated data to file.  */
192     protected void initDataRecorder() {
193         String header = "vec. Deffuant batch model\nRandom seed: " +getRngSeed();
194         recorder = new DataRecorder("./models/vec. DeffuantBatch.txt" , this, header );
195         recorder.createNumericDataSource( " ", this, "getRegionCount" , -1, -1);
196         recorder.createNumericDataSource( " ", this, "getZoneCount" , -1, -1);
197         recorder.createNumericDataSource( " ", this, "getRegionSize" , -1, -1);
198         recorder.createNumericDataSource( " ", this, "getZoneSize" , -1, -1);
199         recorder.createNumericDataSource( " ", this, "getDisagreementCount" , -1, -1);
200     }
201
202     public void setup() {
203
204         schedule = null;
205
206         System.gc();
207
208         schedule = new Schedule(1);
209         agentList = new ArrayList<DeffuantAgent>();
210         space = null;
211         recorder = null;
212     }
213
214     public static void main(String[] args) {
215         SimInit init = new SimInit();
216         DeffuantBatchModel model = new DeffuantBatchModel();
217         init.loadModel(model, null, false);
218     }
219
220     class CountAction extends BasicAction {
221         public void execute() {
222             regionCount = regionCounter.countRegions();
223             zoneCount = regionCounter.countZones();
224             aveZoneSize = regionCounter.aveZonesSize();
225             aveRegionSize = regionCounter.aveRegionSize();
226             disagreementCount = regionCounter.countDisagreements();
227
228             }
229         }
230
231     class Interaction extends BasicAction {
232         public void execute() {
233         boolean event;
234         int bitCount=1;
235             do{
236             int i = Random.uniform.nextIntFromTo(0, agentList.size()-1);     // Colt method.
237             DeffuantAgent agent = (DeffuantAgent) agentList.get(i);
238
239             if(++bitCount < 1000) event =  agent.step(agent);
240             else event = true;
241
242             }while(event = false);
243         }
```

```
244     }
245
246     class Mutation extends BasicAction {
247         public void execute() {
248             if( Random.uniform.nextDoubleFromTo(0, 1) <=  mutationProbability  ) {
249                 DeffuantAgent agent = (DeffuantAgent)
    agentList.get(Random.uniform.nextIntFromTo(0, agentList.size()-1));
250                 agent.mutate();
251             }
252         }
253     }
254
255     class OutputAction extends BasicAction {
256         public void execute() {
257             System.out.println((long)getTickCount()+" ticks: "+regionCount+" regions,
    "+zoneCount+" zones, "+aveZoneSize+" aveZoneSize, "+aveRegionSize+" aveRegionSize,
    "+disagreementCount+" disagreements " );
258         }
259     }
260
261     /*
262         Stops the simulation when the model converges to stable regions (zones)
263         or 100000000 ticks is reached.
264         The final number of regions/zones is output.
265     */
266     class StopAction extends BasicAction {
267         public void execute() {
268             if(regionCount == zoneCount) {
269                 System.out.println((long)getTickCount()+" ticks: "+regionCount+" regions,
    "+zoneCount+" zones " );
270                 long stop = System.currentTimeMillis();
271                 System.out.println("Converged: elapsed time = " +(stop-start)/1000+" secs");
272                 stop();
273                 }
274             else if((long)getTickCount()== 1000000000) {
275                 System.out.println((long)getTickCount()+" ticks: "+regionCount+" regions,
    "+zoneCount+" zones " );
276                 System.out.println("Reached 1000000000 ticks" );
277                 stop();
278                 }
279         }
280     }
281 }
282
```