```
 1
 2 package vecDeffuant;
 3
 4 import java.awt.*;
13
14 //The class DeffuantAgent defines the possible configuration for each site and how
   sites/agents interact with one another.
15
16 public class DeffuantAgent implements Drawable {
17
18     private int featureCount; // Number of cultural features.
19     public int feature;
20     private int threshold;    // the interaction threshold. Must be between 0 and
   featureCount
21     private RegionCounter regionCounter;
22     private double p;       // The probability that the interaction result is in favour  of
   the agent with config 1 when when two agent interact w.r.t. a certain feature
23     private AgentColour siteColour; // Object that determines the colour  of the displayed
   agent.
24     private Grid space;     // The grid in which the agents are situated.
25     private int[] traitCounts;  // The number of traits possessed by each feature. Allows
   for variable numbers of traits, but all are set to two in the vec . Deff. model.
26     private int[] traits;   // The array of current configuration (\in {0,1}^featureCount).
27     private double dissociating; //Corresponds to the kappa in the probability generator of
   the dissociating vec . Deff. Model. Default set to 1.0 (standard vec . Deff. Model)
28
29     /** Indicates whether this site has been processed in analysing  regions and zones.  */
30     protected boolean done; // N.B. accessed directly from RegionCounter class.
31
32     /** Agent's grid coordinates.   */
33     protected int x, y; // N.B. accessed directly from RegionCounter class.
34
35     /**
36         Create the agent.
37
38         @param x, y: the grid coordinates for agent.
39         @param space: the grid in which the agents are situated.
40         @param featureCount: number of cultural features possessed by the agent.
41         @param traitCount: number of traits possessed by all features.
42         @param initialTraits: initial configuration.
43         @param siteColour: the object that determines the colour  of the displayed agent.
44     */
45     public DeffuantAgent(int x, int y, Grid space, int featureCount, int traitCount,
46             int[] initialTraits, AgentColour siteColour, int threshold, double p, double
   dissociating) {
47         this.x = x;
48         this.y = y;
49         this.featureCount = featureCount;
50         this.threshold = threshold;
51         this.p = p;
52         this.dissociating = dissociating;
53         this.space = space;
54         this.siteColour = siteColour;
55         this.traitCounts = new int[featureCount];
56         this.traits = new int[featureCount];
57         for( int i = 0; i < featureCount; i++ ) {
58             this.traits[i] = initialTraits[i];
59             this.traitCounts[i] = 2;
60         }
61     }
62
63     /**
```

```
64          Implements Drawable interface. Agents are drawn with different colors to identify
    their
65          current configuration.
66      */
67      public void draw(SimGraphics g) {
68          Color c = siteColour.getColour(traits);
69          g.drawFastRect(c);
70      }
71
72      // get/set methods allowing the agent's state to be probed.
73      // ------------------------------------------------------
74      public int getX() {return x;}
75      public int getY() {return y;}
76      public String getTraits() {return traitsToString();}
77      public void setTraits(String newTraits) {
78          boolean traitsChanged = false;
79          int i, t;
80          String s;
81          StringTokenizer st = new StringTokenizer( newTraits, " ,\t\n\r\f" );
82          if(st.countTokens() != featureCount ) {
83              System.out.println("Incorrect number of traits." );
84              return;
85          }
86          int[] nt = new int[featureCount];
87          for( i = 0; i < featureCount; i++ ) {
88              s = st.nextToken();
89              try {
90                  t = Integer.parseInt(s);
91                  if( t < 0 || t >= 2 ) {
92                      System.out.println("Configuration "+t+" is invalid.");
93                      return;
94                  }
95                  nt[i] = t;
96              } catch(NumberFormatException ex) {
97                  System.out.println("Configuration "+s+" is invalid.");
98                  return;
99              }
100         }
101         for( i = 0; i < featureCount; i++ ) {
102             if(traits[i] != nt[i]) {
103                 traitsChanged = true;
104                 traits[i] = nt[i];
105             }
106         }
107         if(traitsChanged)
108             System.out.println("Site ("+x+","+y+") config. changed to "+newTraits+".");
109     }
110
111     /**
112         Returns the number of features in this site that are different from the specified
    site.
113         (Used to count regions.) 'Distance' implies the Hamming  distance on the state
    space.
114
115         @param site another site.
116         @return cultural distance.
117         @throws IllegalArgumentException if sites have different numbers of features.
118     */
119     public int distance(DeffuantAgent site) {
120         int count = 0;
121         if(this.featureCount != site.featureCount)
122             throw new IllegalArgumentException( "Incompatible sites." );
```

```
123         for( int i = 0; i < featureCount; i++ )
124             if(this.traits[i] != site.traits[i]) ++count;
125         return count;
126     }
127 
128 
129     public int getTrait(int feature) {
130         if(feature < 0 ||  feature > featureCount)
131             throw new IllegalArgumentException( "feature not in range" );
132         return this.traits[feature];
133     }
134 
135     /**
136         interactStandard() defines interaction btw  two agents in the standard vec . Deff.
    Model
137         Convergent interaction with a neighbouring  site.
138         Select at random a feature on which this agent and its neighbour
139         differ (if there is one) and let a Bernoulli  random variable with density p (p=0.5
    default) decide who
140         assimilates.
141 
142         @param neighbour a neighbouring  agent.
143         @param winner stores the value of the Bernoulli  random variable
144         @param featuresDiffer stores the features which differ between agent and neighbour
    in a list
145         @return true if a change took place; false otherwise.
146 
147     */
148 
149     private boolean interactStandard( DeffuantAgent  neighbour) {
150         int featureTry;      // bit being tried looking for dissimilarity.
151         double winner;
152         ArrayList<Integer>  featuresDiffer = new ArrayList<Integer>();
153 
154         for( int i = 0; i < featureCount; i++ ){
155             if(this.traits[i] != neighbour.traits[i])
156             {featuresDiffer.add(i);}
157         }
158 
159         if(featuresDiffer.isEmpty()){
160             return false;
161         }
162     else{
163 
164         Collections.shuffle(featuresDiffer);
165         featureTry = featuresDiffer.get(0);
166         winner = Random.uniform.nextDoubleFromTo(0,1);
167 
168             if( traits[featureTry] > neighbour.traits[featureTry] ) {
169                 if(winner <= p ){
170                     neighbour.traits[featureTry] = traits[featureTry];
171                     } else{
172                 traits[featureTry] = neighbour.traits[featureTry];
173                     }
174                 return true;
175                 }
176             else{
177                 if(winner <= p ){
178                     traits[featureTry] = neighbour.traits[featureTry];
179                         } else{
180                             neighbour.traits[featureTry] = traits[featureTry];
181                     }
```

```
182                 return true;
183             }
184       }
185     }
186
187     /**
188     interactDissociating() defines interaction btw  two agents in the dissociating vec .
    Deff. Model
189     Convergent interaction with a neighbouring  site.
190     Select at random a feature on which this agent and its neighbour
191     differ (if there is one). Which agent has to assimilate now depends on their current
    configuration
192     and a Bernoulli  random variable with density q (p=dissociating/(1+dissociating)).
193     Several cases have to be checked to make interaction correspond to the flip rates of
    the diss. vec. Deff. Model
194
195     @param agent a randomly selected agent.
196     @param neighbour a neighbouring  agent.
197     @param winner stores the value of the Bernoulli  random variable
198     @param featuresDiffer stores the features which differ between agent and neighbour   in a
    list
199     @return true if a change took place; false otherwise.
200
201 */
202
203     private boolean interactDissociating(DeffuantAgent  agent, DeffuantAgent neighbour) {
204         int bitTry;     // bit being tried looking for dissimilarity.
205         double winner;
206         DeffuantAgent  zeros;
207         ArrayList<Integer>  featuresDiffer = new ArrayList<Integer>();
208         for( int i = 0; i < featureCount; i++ ){
209             if(this.traits[i] != neighbour.traits[i])
210             {featuresDiffer.add(i);}
211         }
212         int[] zeroTraits = new int[featureCount];
213         for(int i = 0; i < featureCount; i++) zeroTraits[i] = 0;
214         zeros = new DeffuantAgent(-1, -1,  null, featureCount, 2, zeroTraits, null,
    threshold, p, dissociating);
215
216         double centralPosition = (double) featureCount/2;
217         double agentPosition = agent.distance(zeros);
218         double neighbourPosition = neighbour.distance(zeros);
219         double q = dissociating/(1+dissociating);
220
221     if( !featuresDiffer.isEmpty()){
222      if(agentPosition <= centralPosition && neighbourPosition<=centralPosition &&
    agentPosition < neighbourPosition){
223
224             Collections.shuffle(featuresDiffer);
225             bitTry = featuresDiffer.get(0);
226             winner = Random.uniform.nextDoubleFromTo(0,1);
227
228             if(traits[bitTry] > neighbour.traits[bitTry] ) {
229                 if(winner <= 0.5){
230                     neighbour.traits[bitTry] = traits[bitTry];
231                     } else{
232                 traits[bitTry] = neighbour.traits[bitTry];
233                     }
234                 return true;
235                 }
236             else{
237                 if(winner <= 1-q){
```

```
238                     traits[bitTry] = neighbour.traits[bitTry];
239                         } else{
240                             neighbour.traits[bitTry] = traits[bitTry];
241                         }
242                     return true;
243                 }
244         }
245             if(agentPosition <= centralPosition && neighbourPosition<=centralPosition &&
    agentPosition > neighbourPosition ){
246
247                     Collections.shuffle(featuresDiffer);
248                     bitTry = featuresDiffer.get(0);
249                     winner = Random.uniform.nextDoubleFromTo(0,1);
250
251                         if(traits[bitTry] > neighbour.traits[bitTry] ) {
252                             if(winner <= 1-q){
253                                 neighbour.traits[bitTry] = traits[bitTry];
254                                 } else{
255                             traits[bitTry] = neighbour.traits[bitTry];
256                                 }
257                             return true;
258                             }
259                         else{
260                             if(winner <= 0.5){
261                                 traits[bitTry] = neighbour.traits[bitTry];
262                                     } else{
263                                         neighbour.traits[bitTry] = traits[bitTry];
264                                 }
265                             return true;
266                         }
267                 }
268             if(agentPosition >= centralPosition && neighbourPosition >=centralPosition &&
    agentPosition < neighbourPosition ){
269
270                     Collections.shuffle(featuresDiffer);
271                     bitTry = featuresDiffer.get(0);
272                     winner = Random.uniform.nextDoubleFromTo(0,1);
273
274                         if(traits[bitTry] > neighbour.traits[bitTry] ) {
275                             if(winner <= 0.5){
276                                 neighbour.traits[bitTry] = traits[bitTry];
277                                 } else{
278                             traits[bitTry] = neighbour.traits[bitTry];
279                                 }
280                             return true;
281                             }
282                         else{
283                             if(winner <= q){
284                                 traits[bitTry] = neighbour.traits[bitTry];
285                                     } else{
286                                         neighbour.traits[bitTry] = traits[bitTry];
287                                 }
288                             return true;
289                         }
290                 }
291                 if(agentPosition >= centralPosition && neighbourPosition >=centralPosition &&
    agentPosition > neighbourPosition ){
292
293                     Collections.shuffle(featuresDiffer);
294                     bitTry = featuresDiffer.get(0);
295                     winner = Random.uniform.nextDoubleFromTo(0,1);
296
```

```
297                         if(traits[bitTry] > neighbour.traits[bitTry] ) {
298                             if(winner <= q){
299                                 neighbour.traits[bitTry] = traits[bitTry];
300                                 } else{
301                             traits[bitTry] = neighbour.traits[bitTry];
302                                 }
303                             return true;
304                             }
305                         else{
306                             if(winner <= 0.5){
307                                 traits[bitTry] = neighbour.traits[bitTry];
308                                     } else{
309                                         neighbour.traits[bitTry] = traits[bitTry];
310                                 }
311                             return true;
312                         }
313                 }
314             else{
315 
316              Collections.shuffle(featuresDiffer);
317              bitTry = featuresDiffer.get(0);
318              winner = Random.uniform.nextDoubleFromTo(0,1);
319 
320              if(traits[bitTry] > neighbour.traits[bitTry] ) {
321                  if(winner <= 0.5 ){
322                      neighbour.traits[bitTry] = traits[bitTry];
323                      } else{
324                  traits[bitTry] = neighbour.traits[bitTry];
325                      }
326                  return true;
327                  }
328              else{
329                  if(winner <= 0.5 ){
330                      traits[bitTry] = neighbour.traits[bitTry];
331                          } else{
332                              neighbour.traits[bitTry] = traits[bitTry];
333                      }
334                  return true;
335              }
336          }
337      }
338      else {return false;}
339  }
340 
341      /**
342          Returns the number of characters in an integer number.
343          The sign of the number is included.
344 
345          @return the length of n.
346      */
347      private static int lengthOf(int n)
348          { return(Integer.toString(n).length()); }
349 
350      /** Performs a mutation. A features configuration is randomly changed.  */
351      public void mutate() {
352          int oldAllele, newAllele, bit, allelemax;
353 
354          bit = Random.uniform.nextIntFromTo(0, featureCount-1);    // Randomly choose a
    feature to mutate.
355          oldAllele = traits[bit];
356          allelemax = 2;
357          do  {
```

```
358             newAllele = Random.uniform.nextIntFromTo(0, allelemax-1);
359         } while(newAllele == oldAllele);
360         traits[bit] = newAllele;
361     }
362 
363     /**
364         Executes one step of the RePast simulation.
365         This method only attempts one interaction event,  which may not require updating the
    display etc.
366         Within one tick the RePast simulation will attempt up to 10000 steps until an
    interaction is successful.
367 
368         @return true if an interaction took place; false otherwise.
369     */
370     public boolean step(DeffuantAgent agent) {
371         DeffuantAgent neighbour;
372         neighbour = (DeffuantAgent) space.getNeighbour(x,y);     // Randomly choose a
    neighbour.
373 
374             if(dissociating == 1){
375         if(this.distance(neighbour) <= threshold && this.distance(neighbour) != 0)
376             {return(interactStandard( neighbour));}
377         else return false;
378             }
379             else{
380         if(this.distance(neighbour) <= threshold && this.distance(neighbour) != 0)
381             {return(interactDissociating( agent,  neighbour));}
382         else return false;
383             }
384     }
385 
386     /** Return the current configuration as a string of numbers.     */
387     public String traitsToString() {
388         StringBuffer sb = new StringBuffer(featureCount*2);
389         for( int i = 0; i < featureCount; i++ ) {
390             for( int j = 0; j < lengthOf(1)-lengthOf(traits[i]); j++ )
391                 sb.append(" ");
392             sb.append(traits[i]+" ");
393         }
394         return sb.toString();
395     }
396 }
397 
398 
399 
```