```
1)     public Measure[] crossover(Measure measure1, Measure measure2)
2)     {
3)           Random random = new Random(System.nanoTime());
4)           int pointOfCross = 0;
5)           ArrayList<Integer> possibleCrossPoints = new ArrayList<Integer>();
6)
7)           //find the acceptable points of crossover
8)           for (int i = 0; i < tickLength; i++)
9)           {
10)                if (!measure1.isNoteOnAtTime(i) && !measure2.isNoteOnAtTime(i))
11)                           possibleCrossPoints.add(i);
12)          }
13)
14)          if (possibleCrossPoints.size() < 1) //cannot preform crossover
15)          {
16)                Measure[] result = new Measure[2];
17)                result[0] = measure1;
18)                result[1] = measure2;
19)                return result;
20)          }
21)
22)     pointOfCross = possibleCrossPoints.get(random.nextInt(possibleCrossPoints.size()));
23)
24)          ArrayList<Note> temp1 = new ArrayList<Note>();
25)          ArrayList<Note> temp2 = new ArrayList<Note>();
26)
27)          for (int i = 0; i < measure1.getNotes().size(); i++)
28)          {
29)                if (measure1.getNotes().get(i).getTime() > pointOfCross)
30)                {
31)                      temp1.add(measure1.getNotes().get(i));
32)                      measure1.removeNoteAt(i--);
33)                }
34)          }
35)
36)          for (int i = 0; i < measure2.getNotes().size(); i++)
37)          {
38)                if (measure2.getNotes().get(i).getTime() > pointOfCross)
39)                {
40)                      temp2.add(measure2.getNotes().get(i));
41)                      measure2.removeNoteAt(i--);
42)                }
43)          }
44)
45)          measure1.addNotes(temp2);
46)          measure2.addNotes(temp1);
47)
48)          Measure[] result = new Measure[2];
49)          result[0] = measure1;
50)          result[1] = measure2;
51)          return result;
52)    }
53)
54)    public Measure pointMutation(Measure measure)
55)    {
56)          if (measure.getNotes().size() == 0)
57)                return createRandomMeasure();
58)
```

```
59)         Random random = new Random(System.nanoTime());
60)         int amountToChange = random.nextInt(measure.getNotes().size());
61)         int rnd = random.nextInt(measure.getNotes().size());
62)         int[] used = new int[measure.getNotes().size()];
63)         int temp = 0;
64)
65)         for (int i = 0; i < amountToChange; i++)
66)         {
67)               while (Utility.searchArrayForInteger(rnd, used))
68)               {
69)                     rnd = random.nextInt(measure.getNotes().size());
70)               }
71)
72)              measure.getNotes().get(rnd).setKey((byte) (random.nextInt(24) - 12 +
                 octave));
73)               used[temp++] = rnd;
74)         }
75)
76)         return measure;
77)   }
78)   //this mutation will split a duration into 2 new notes
79)   public Measure durationSplit(Measure measure)
80)   {
81)         Random random = new Random(System.nanoTime());
82)         int amountToChange = random.nextInt(measure.getNotes().size());
83)         int rnd = random.nextInt(measure.getNotes().size());
84)         int[] used = new int[measure.getNotes().size() * 2];
85)         int temp = 0;
86)
87)         for (int i = 0; i < amountToChange; i++)
88)         {
89)               while (Utility.searchArrayForInteger(rnd, used))
90)               {
91)                     rnd = random.nextInt(measure.getNotes().size());
92)               }
93)
94)               measure.splitNote(rnd);
95)               used[temp++] = rnd;
96)               used[temp++] = rnd + 1;
97)
98)               //update used positions if they are greater since we are adding notes
99)               for (int j = 0; j < used.length; j++)
100)              {
101)                    if (used[j] > rnd + 1)
102)                          used[j]++;
103)              }
104)        }
105)
106)        return measure;
107)  }
108)
109)  //The random measure generation selects 2 random durations, then randomly adds them
      to the track as a note or pause
110)  //the generator also randomly selects a key for the note, + or - 12 from the octave
111)  public Measure createRandomMeasure()
112)  {
113)        Random random = new Random(System.nanoTime());
114)        Measure measure = new Measure(0, timeDivision);
```

```
115)
116)         int duration1 = Utility.getDurationFromString(timeDivision,
     Definitions.NOTE_LENGTHS[random.nextInt(Definitions.NOTE_LENGTHS.length)] + "");
117)
118)         int duration2 = Utility.getDurationFromString(timeDivision,
     Definitions.NOTE_LENGTHS[random.nextInt(Definitions.NOTE_LENGTHS.length)] + "");
119)
120)         int totalDuration = 0;
121)
122)         while (totalDuration != tickLength)
123)         {
124)                 random = new Random(System.nanoTime());
125)
126)                 //Note on
127)                 if (random.nextInt(4) <= 2)
128)                 {
129)                         byte key = (byte) (random.nextInt(24) - 12 + octave);
130)
131)                         //first duration
132)                         if (random.nextInt(2) == 1 && totalDuration + duration1 <=
                             tickLength)
133)                         {
134)                                 measure.addNote(new Note(channel, key, totalDuration,
                                     duration1, (byte)127));
135)                                 totalDuration += duration1;
136)                         }
137)                         //second duration
138)                         else if (totalDuration + duration2 <= tickLength)
139)                         {
140)                                 measure.addNote(new Note(channel, key, totalDuration,
                                     duration2, (byte)127));
141)                                 totalDuration += duration2;
142)                         }
143)                         else
144)                                 totalDuration = tickLength;
145)                 }
146)                 //note off
147)                 else
148)                 {
149)                         //first duration
150)                         if (random.nextInt(2) == 1 && totalDuration + duration1 <=
                             tickLength)
151)                         {
152)                                 totalDuration += duration1;
153)                         }
154)
155)                         else if (totalDuration + duration2 <= tickLength)
156)                         {
157)                                 totalDuration += duration2;
158)                         }
159)                         else
160)                                 totalDuration = tickLength;
161)                 }
162)                 if (totalDuration > tickLength)
163)                 {
164)                         measure = new Measure(0, timeDivision);
165)                 }
166)         }
```

```
        checkForHarmonies(measure);
        return measure;
}
```