1. Melody Generation with Markov Chains

Markov chains, named after Andrey Markov, are mathematical systems that hop from one "state" (a situation or set of values) to another. For example, if you made a Markov chain model of a baby's behavior, you might include "playing," "eating", "sleeping," and "crying" as states, which together with other behaviors could form a 'state space'. In Music Theory arcade, Markov Chains are widely used to generate automatic random melody progressions especially present in the #2 Game – Piano Repeater. Each note has a transition state which could be possible in given scale. For example, in C – Major, the possible states are: "C" → {"CC", "CD", "CE", "CF", "CG", "CA", "CB"}. As known from the Music Theory the possible notes in the C – Major are C, D, E, F, G, A, B. In the melodyGeneration.pde first the Chromatic Scale is defined:

```
1.  public static String Chromatic_Scale[]={"C", "Cs", "D", "Ds",
    "E", "F", "Fs", "G", "Gs", "A", "As", "B", "C", "Cs", "D",
    "Ds", "E", "F", "Fs", "G", "Gs", "A", "As", "B"};
```

As also mentioned during the lectures the Major scale is defined by the intervals between these notes starting from the root note by: W W H W W W H. W indicates the whole step while H indicates the Half-Step. On the other hand, minor scale is defined as: W H W W H W W. The program simply defines scales according to those formulas given above.

```
1.      //Major Scale W-W-H-W-W-W-H (W: Wholestep, H: Halfstep):
2.       noteStates[0] = Chromatic_Scale[0];
3.       noteStates[1] = Chromatic_Scale[0+2];
4.       noteStates[2] = Chromatic_Scale[0+2+2];
5.       noteStates[3] = Chromatic_Scale[0+2+2+1];
6.       noteStates[4] = Chromatic_Scale[0+2+2+1+2];
7.       noteStates[5] = Chromatic_Scale[0+2+2+1+2+2];
8.       noteStates[6] = Chromatic_Scale[0+2+2+1+2+2+2];
```

If we convert the formula for major scale, say W is equal to 2 and H is equal to 1. So one can define them easily in any programming language. Here in the0 corresponds to the note C so 0 + 2 → D, 0 + 2 + 2 → E, 0 + 2 + 2 + 1 → F. The melody generation is then moves into the next stage where notes are moved next to each other with respect to their probability distribution from online available databases mainly regarding to EDM.

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
C	0.2		0.2	0.1		0.1		0.2	0.1		0.1	
C#												
D	0.1		0.1	0.2		0.2		0.2	0.1		0.1	
D#	0.2		0.2	0.1		0.2		0.1	0.1		0.1	
E												
F	0.2		0.2	0.1		0.1		0.2	0.1		0.1	
F#												
G	0.2		0.2	0.1		0.2		0.1	0.1		0.1	
G#	0.1		0.2	0.1		0.1		0.2	0.1		0.2	
A												
A#	0.1		0.2	0.1		0.1		0.3	0.1		0.1	
B												
C Minor												

Figure 1: C - Minor Transition State Table

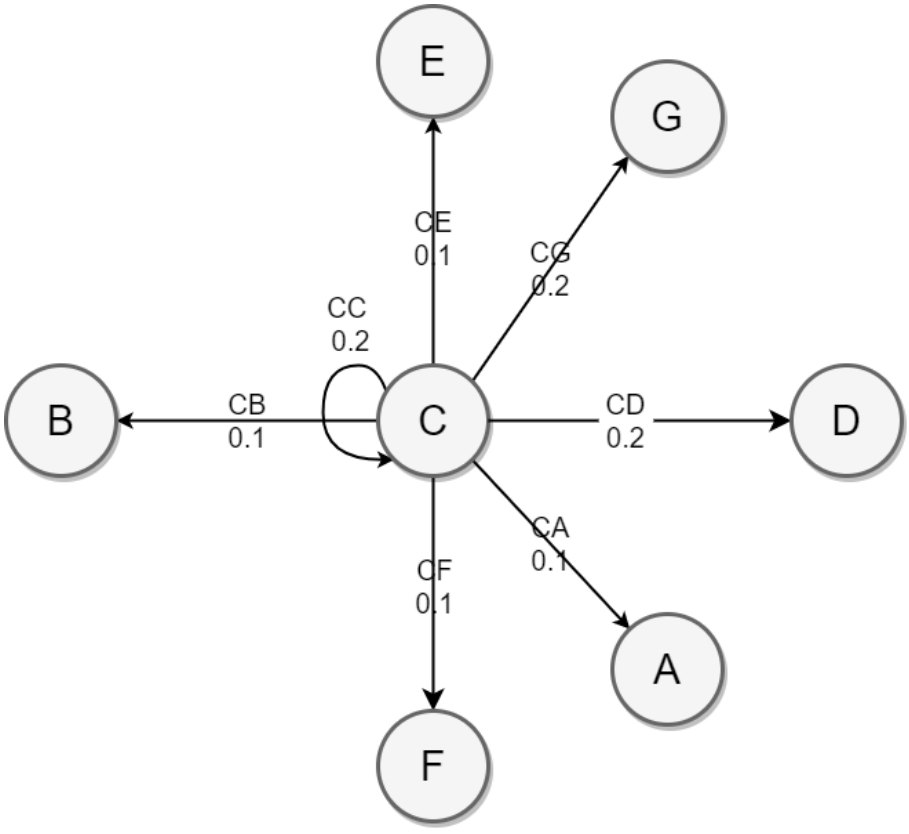

Figure 2: C- Major, Possible States for Note C

```
switch(scale) {
  case "Major":
    // Switch-Case Structure for Root Note Selection
    switch(noteKey) {
    case "C":
      //Major Scale W-W-H-W-W-W-H (W: Wholestep, H: Halfstep):
      noteStates[0] = Chromatic_Scale[0];
      noteStates[1] = Chromatic_Scale[0+2];
      noteStates[2] = Chromatic_Scale[0+2+2];
      noteStates[3] = Chromatic_Scale[0+2+2+1];
      noteStates[4] = Chromatic_Scale[0+2+2+1+2];
      noteStates[5] = Chromatic_Scale[0+2+2+1+2+2];
      noteStates[6] = Chromatic_Scale[0+2+2+1+2+2+2];

      // Starting State:
      b = rnd.nextInt(((7-1)-0)+1)+0;
      startNote = noteStates[b];
      easyMelody[0] = startNote;
      print("Start Note:", startNote);
      print("\n");

      // Markov Chain Calculation to determine next states with respect to probability
      while (i != difficultySelection) {
        if (startNote == noteStates[0])
        {
          // Transition States from note "C" ---> {"CC", "CD", "CE", "CF", "CG", "CA", "CB"}
          EnumeratedIntegerDistribution dist = new EnumeratedIntegerDistribution(new int[]{0, 1, 2, 3, 4, 5, 6}, new double[]{0.2, 0.2, 0.1, 0.1, 0.2, 0.1, 0.1});
          int idx = dist.sample();
          easyMelody[i] = noteStates[idx];
          startNote = noteStates[idx];
        } else if (startNote == noteStates[1])
        {
          // Transition States from note "D" ---> {}
          EnumeratedIntegerDistribution dist = new EnumeratedIntegerDistribution(new int[]{0, 1, 2, 3, 4, 5, 6}, new double[]{0.1, 0.1, 0.2, 0.2, 0.2, 0.1, 0.1});
          int idx = dist.sample();
          easyMelody[i] = noteStates[idx];
          startNote = noteStates[idx];
        } else if (startNote == noteStates[2])
        {
          // Transition States from note "Ds" ---> {}
          EnumeratedIntegerDistribution dist = new EnumeratedIntegerDistribution(new int[]{0, 1, 2, 3, 4, 5, 6}, new double[]{0.2, 0.2, 0.1, 0.2, 0.1, 0.1, 0.1});
          int idx = dist.sample();
          easyMelody[i] = noteStates[idx];
          startNote = noteStates[idx];
        } else if (startNote == noteStates[3])
        {
          // Transition States from note "F" ---> {}
```

```
48.           EnumeratedIntegerDistribution dist = new EnumeratedIntegerDistribution(new
    int[]{0, 1, 2, 3, 4, 5, 6}, new double[]{0.2, 0.2, 0.1, 0.1, 0.2, 0.1, 0.1});
49.           int idx = dist.sample();
50.           easyMelody[i] = noteStates[idx];
51.           startNote = noteStates[idx];
52.         } else if (startNote == noteStates[4])
53.         {
54.           // Transition States from note "G" ---> {}
55.           EnumeratedIntegerDistribution dist = new EnumeratedIntegerDistribution(new
    int[]{0, 1, 2, 3, 4, 5, 6}, new double[]{0.2, 0.2, 0.1, 0.2, 0.1, 0.1, 0.1});
56.           int idx = dist.sample();
57.           easyMelody[i] = noteStates[idx];
58.           startNote = noteStates[idx];
59.         } else if (startNote == noteStates[5])
60.         {
61.           // Transition States from note "Gs" ---> {}
62.           EnumeratedIntegerDistribution dist = new EnumeratedIntegerDistribution(new
    int[]{0, 1, 2, 3, 4, 5, 6}, new double[]{0.1, 0.2, 0.1, 0.1, 0.2, 0.1, 0.2});
63.           int idx = dist.sample();
64.           easyMelody[i] = noteStates[idx];
65.           startNote = noteStates[idx];
66.         } else if (startNote == noteStates[6])
67.         {
68.           // Transition States from note "As" ---> {}
69.           EnumeratedIntegerDistribution dist = new EnumeratedIntegerDistribution(new
    int[]{0, 1, 2, 3, 4, 5, 6}, new double[]{0.1, 0.2, 0.1, 0.1, 0.3, 0.1, 0.1});
70.           int idx = dist.sample();
71.           easyMelody[i] = noteStates[idx];
72.           startNote = noteStates[idx];
73.         }
74.         // Iterate the Loop Variable
75.         i++;
76.       }
77.       // Break the Root Note = C-Minor case
78.       break;
```