Example of notes and the Mechbass note range:

1. *@97680(SHORT\_MESSAGE) Channel 15: Note on, D2 key=38 velocity: 95  
   @97800(SHORT\_MESSAGE) Channel 15: Note off, D2 key=38 velocity: 80*
2. @97800(SHORT\_MESSAGE) Channel 15: Note on, ASharp2 key=46 velocity: 95  
   @97920(SHORT\_MESSAGE) Channel 15: Note off, ASharp2 key=46 velocity: 80
3. *@97920(SHORT\_MESSAGE) Channel 15: Note on, E1 key=28 velocity: 95  
   @98040(SHORT\_MESSAGE) Channel 15: Note off, E1 key=28 velocity: 80*
4. @98040(SHORT\_MESSAGE) Channel 15: Note on, E1 key=28 velocity: 95  
   @98160(SHORT\_MESSAGE) Channel 15: Note off, E1 key=28 velocity: 80
5. *@98160(SHORT\_MESSAGE) Channel 15: Note on, D2 key=38 velocity: 95*

*@98280(SHORT\_MESSAGE) Channel 15: Note off, D2 key=38 velocity: 80*

1-2-3-4-5

\*\*MECHBASS NOTE RANGE\*\*

|  |  |  |  |
| --- | --- | --- | --- |
|  | **MIDI CHANNEL** | **LOWEST NOTE** | **HIGHEST NOTE** |
| **G STRING** | 1 | 43 | 56 |
| **D STRING** | 2 | 38 | 51 |
| **A STRING** | 3 | 33 | 46 |
| **E STRING** | 4 | 28 | 41 |

(difference of 13 between lowest and highest note)

NOTES ON MIDI NOTE NUMBERS:

Midi note numbers go from 0 to 127

0 is octave -2 note C

127 is octave 9 note G

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Node:

private ShortMessage note = <object containing all information about the note>;

private long tick = <when the node’s ShortMessage occured>;

private List<Node> neighbours = <a list of all the nodes connected to this one>;

private List<integer> strings = <a list of integers ranging from 0 – n-1, representing the number of strings the note can fit on (e.g. if this node has a note of 40, the list will be [2, 3, 4]. if this node has a note of 52, the list will be [1]. Refer to the table above for the note ranges that fit on which string)>;

private int stringToPlayOn = Integer.MIN\_VALUE; //<an integer ranging from 0 – n-1, representing the string the note will be played on (n being the number of strings, unless it's on a dummy string)> - this starting value is a flag that the string is unassigned.

CONSTRUCTOR

GETTERS

SETTERS

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Mock Pseudocode:**

**Takes 2 parameters: a Sequence object, seq and a MekString[] array object, str and Returns a Sequence.**

**//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//\\//**// Initialise the nodes with the short messages

ArrayList<Node> allNodes = <New list>;

strings = str;

Track tr = seq.getTracks()[0];

// loop through

for (int i=0; i<tr.size(); ++i){

MidiMessage midmsg = tr.get(i).getMessage();

// check what type of message it is

if (midmsg instanceof ShortMessage){

ShortMessage shrtmsg = (ShortMessage) midmsg;  
 // Add a new node, then set the recently created node’s ShortMessage to the recently created shrtmsg.

Node newNode = new Node();

allNodes.add(newNode);

newNode.setShortMessage(shrtmsg);

// Get the tick of the event attached to the message.

newNode.setTick(tr.get(i).getTick);

// Now go through the strings and check if the note can be played on any of them.

int note = shrtmsg.getData1();

for (int j = 0; j < j++) {

if (note >= strings[j].lowNote && note <= strings[j].highNote)

newNode.addPossibleString(j);

}

}

}

For(int j = 1; j < allNodes.size; j++){

For(int n = 0; n < j; n++){

Node node1 = allNodes.get(j);

Node node2 = allNodes.get(n);

For (int o = 0; o < strings.length; o++){

If (node1.getPossibleStrings().contains(o) && node2.getPossibleStrings().contains(o)){

If (strings[o].conflicting(node1.getNote(),node2.getNote(), node1.getTick() - node1.getTick()) && (!node1.getNeighbours().contains(node2) || !node2.getNeighbours().contains(node1)) ){

Node1.addNeighbour(node2);

Node2.addNeighbour(node1);

}

}

}

}

}

// Now that the nodes are initialised, we can move onto using this graph to assign strings.

For(int j = 0; j < allNodes.size; j++){

Node node = allNodes.get(j);

If (node.getStringToPlayOn() != Integer.MIN\_VALUE || node.getPossibleStrings().isEmpty())

Continue;

Else{

For (Node n : node.getNeighbours()){

For (Integer i : node.getPossibleStrings()){

If (!n.getPossibleStrings().contains(i) || (n.getPossibleStrings().contains(i) && node.getPossibleStrings().size == 1))

Node.setString(i);

}

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

UNFINISHED.

http://i.stanford.edu/pub/cstr/reports/cs/tr/80/830/CS-TR-80-830.pdf

ALGORITHM 6-COLOR.

Given an n vertex planar graph G in adjacency list form, this algorithm

determines a 6-coloring of G.

Step 1. [Establish degree lists.]

For each j where 0 <= j <= (n-1), form a doubly linked list of all vertices of G of degree j.

Step 2. [Label vertices smallest degree last.]

For i=n, n-1, n-1, ... , 1 designate the first vertex of the non-vacuous j degree list of

smallest j as vertex t/i. Delete vi from the j degree list. For each vertex U’ that was adjacent to

tli in G and remains in some degree list, say f, delete - u’ from the jr degree list and insert u’

in the j9 - 1 degree list.

Step 3. [Color vertices.]

For i = 1,2,. . . , n, assign vertex t)i the smallest color value (which must be some integer

between one and six) not occuring on the vertices adjacent to t)i that have already been

colored.