CSC 220 Object Oriented Multimedia Programming, Fall 2020

Dr. Dale E. Parson, Assignment 3, MIDI 3D automated avatars in 3D spaces in Processing.

Assignment is due via **D2L Assignments** **CSC220F20Assn3_3DMIDI** by **11:59 PM on November 14**. There will be an in-class work session the first week of November.

Follow the instructions in our Assignment 1[1] for setting up your sketchbook location or download the Processing environment if you need to do this. You probably don't need to do this.

Save a copy of **YOUR Assignment 2 solution** sketch from Processing as **CSC220F20Assn3_3DMIDI** by running File -> Save As -> **CSC220F20Assn3_3DMIDI**. You will copy **SOME** code from my handout file. Specifically:

1. Copy my CollisionDetector class. Copy my Professor class only if you kept it in Assignment 2. I have made changes to CollisionDetector that must replace your CollisionDetector class.

2. Replace your Furniture, Paddle, and VectorAvatar classes and the keyPressed() function. Include all STUDENT comments in these copies and with CollisionDetector.

   KEEP THESE FROM YOUR ASSIGNMENT 2 SKETCH:
   The setup() function requires one line of hand-added code and some added arguments for constructor calls, and draw() receives no changes, so keep your Assignment 2 copies. Do not over-write those. Keep and enhance your custom students avatar class. You need to add the following global variables below to the sketch, along with adding the MIDI Tab. Keep the overlap(…), signum(…), moveCameraRotateWorldKeys(), and makeCustomPShape(...) functions and interface Avatar as they are in your Assignment 2 code (don't copy mine).

   NEW GLOBALS:

```
// Added 10/31/2020 for MIDI display().
int [][] scales = {
  {0, 2, 4, 7, 9, 12}, // major pentatonic -- should give fairly consonant combinations
  {0, 3, 5, 7, 10}, // minor pentatonic
  {0, 2, 4, 5, 7, 9, 11, 12}, // major scale
  {0, 2, 3, 5, 7, 8, 10, 12}  // harmonic minor
 // musician students can add others
} ;
int curscale = 1 ;  // one of the above, LEFT and RIGHT arrows adjust this
int tonic = 0 ;  // offset into above scales, this is the key of C
int octave = 5 ; // 5 * 12 notes in an octave gives us middle C
int volume = 64 ;  // use UP and DOWN ARROWS to adjust overall volume (not per-note velocity)
// for each MIDI channel 0 through 15
```

3. We will go over each of the required STUDENT enhancements on November 3. These are from instruction comments in the code. They are all in the main sketch; there are no changes to the MIDI Tab. You can add additional source code tabs if that is your practice. Make sure to turn in their .pde files if you do.

[1] https://faculty.kutztown.edu/parson/fall2020/csc220fall2020assn1.pdf

```
// STUDENT A 5%: Call initMIDI() before constructing Avatar objects.

// STUDENT B 10%: Classes Professor and VectorAvatar constructors take 3 new
arguments:
//    CHANNEL (0..15), PATCH (PROGRAM_CHANGE), and true|false on whether to us
//    the stereo Balance CONTROL_CHANGE to move sound Left-to-Right per object's
//    x location. Furniture and Paddle just take CHANNEL & PATCH.
//    I gave the first two Professor objects separate channels & stereo balance.
//    I gave the other little Professors the same channel and patch for a chorus.
//    I have some Furniture the same channel & patch, others channel = -1 for no
sound.
//    I gave the Paddles distinct channels. You can make changes to these assignments.

void keyPressed() {
// STUDENT C 15%: implement these global variable changes, making sure not
// to go outside valid value ranges.
//  LEFT and RIGHT adjust the curscale, e.g., major pentatonic.
//  UP and DOWN adjust the global volume on every channel 0 through 15.
// See volume CONTROLLER at
// http://midi.teragonaudio.com/tech/midispec.htm has the CONTROL_CHANGES

// STUDENT D 5%: Copy the changed CollisionDetector into your sketch.
// CollisionDetector has added data fields, constructor parameters,
// and constructor statements in support of MIDI in subclasses, which must now
// supply int MIDIchan and MIDIinstrument arguments in their constructors'
// super(...) calls. Data fields channel and instrument are then available
// to subclasses. CollisionDetector sends the PROGRAM_CHANGE for instrument.
// You must copy CollisionDetector into assn3, over-writing the old one.

// STUDENT E 30%: Integrate channel, instrument, and your own selected
// CONTROL_CHANGE effects into your custom avatar class. Make sure to use
// isStereo on one or two of your objects.
// MIDI OUTPUT FOR PROFESSOR.
// STUDENT - DO YOUR OWN TIMING AND FX IN YOUR AVATAR CLASS.

// STUDENT F 10%: Copy & paste this Furniture class into your sketch,
// then make its display() function play a continuous drone note --
// always the same, low-octave note -- replaying the note every 16
// seconds with no NOTE_OFF.

// STUDENT G 10%: Make the Paddle objects play a drone similar
// to Furniture, but scale the velocity via
// velocity = constrain(int((map(pixwidth, 0.0, width, 0.0, 64.0))),0,64);
// I did int firstPaddleChannel = -1 ;
// and did the above map only on that Paddle, kept the other
// at lower velocity. I also sent a NOTE_ON in every 2 milliseconds.

// STUDENT H 15%: Add stereo to a VectorAvatar.display() when isStereo is true,
// similar to Professor. Also add a CONTROL_CHANGE effect to display().
```

The PORTIONS of modified code that you must copy are here: http://faculty.kutztown.edu/parson/fall2020/CSC220F20Assn3_3DMIDI.txt (handout code). The MIDI code Tab is here. http://faculty.kutztown.edu/parson/fall2020/MIDI.txt You must copy & save all of it as your sketch's MIDI tab.

http://midi.teragonaudio.com/tech/midispec.htm documents the CONTROL_CHANGES.
To figure out your PATCHES and CONTROL CHANGES (effects), experiment running sketch https://faculty.kutztown.edu/parson/fall2020/ConcentricCirclesIntervals.txt. It has its own MIDI Tab linked on the course page.

Search and read **STUDENT** comments in the code before starting.

We will use the one or two classes in the first week of November for working on this project. If you do not get it done in class, you will have to complete it as homework. I expect it to be to me by the due date via D2L. **I will deduct 10% for each day it is late**. Also, re-read the above requirements when you turn it in, to ensure that you don't miss anything. If you make changes after turning it in, just turn in another copy of your sketch via D2L. I will look at the last one that you turn in.

TURNING IT IN: When your work is completed, and you have re-read and satisfied the project requirements, you can use the Windows Explorer to find the file **CSC220F20Assn3_3DMIDI.pde** in your sketch folder. Drag **CSC220F20Assn3_3DMIDI.pde** into the **Assignment 2 dropbox** under our course's D2L account by the due date. If you find you have created an error, you can drop an updated CSC220F20Assn3_3DMIDI.pde into the dropbox. **Assignment 3** is under **Assessments -> Assignments** in our D2L account. If you are working on a laptop or your machine at home, turn it in via D2L in the same way. Also turn in your **MIDI.pde** file if you make changes to it.. Finally, if your sketch loads any image (e.g., for texture) or .svg vector graphics files, and/or if you use Processor editor TABS to create multiple .pde files, turn in those individual files via D2L. I cannot run a sketch that depends on additional files without receiving those files.

Notes below added November 1.

There are only four kinds of MIDI ShortMessages required by this project.

sendMIDI(ShortMessage.PROGRAM_CHANGE, channel, instrument, 0);

The PROGRAM_CHANGE message call appears in the constructor for an object derived from class CollisionDetector. As long as you over-write your Assignment 3 CollisionDetector class with mine, and wire up the subclass (a.k.a. derived class) constructor calls to it via super(…), you should not send PROGRAM_CHANGE in your code unless you decide to play two different instruments on two different channels. This call uses the instrument argument passed to the avatar constructors up in setup().

sendMIDI(ShortMessage.CONTROL_CHANGE, channel, balanceController, balance);

CONTROL_CHANGE calls appear in the avatar class display() functions. All sendMIDI calls in my handout display() functions take place after the graphics calls, after the final pop(), which does not affect MIDI.

sendMIDI(ShortMessage.NOTE_OFF, channel, lastNotePitch, 0);

NOTE_OFF is needed for display()s that change pitch and need to silence the previous note before playing the next one. For droning avatars such as Paddle that keep a single note but change only velocity but not pitch, my code skips the NOTE_OFF and simply plays NOTE_ON with the same pitch but a lower or higher velocity, once every 2 milliseconds.

sendMIDI(ShortMessage.NOTE_ON, channel, lastNotePitch, lastNoteVelocity);

NOTE_ON is for playing a note on this avatar's channel after updating the lastNotePitch and lastNoteVelocity data fields inside the class' data field. See class Professor for an example.

Note that function sendMIDI(…) inside the MIDI Tab validates the channel argument before sending a ShortMessage to the synthesizer, thereby allowing avatar objects constructor with a channel value of -1 to remain silent. My code uses that approach for some of the Furniture objects to avoid aural clutter. The MIDI Tab also defines function getMilliseconds() that returns the number of thousandths of a second since the sketch started running, useful as a note timer per Professor.display().

Constant **final int midiDeviceIndex = 0 ;** in the MIDI Tab should work on your machine. It normally points to the default Java MIDI library software synthesizer.