CSC 220 Object Oriented Multimedia Programming, **Fall 2021**

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

This is a demo for the fall 2022 class, not an assignment.

~~Assignment is due via **D2L Assignments** **CSC220F21Assn3_MIDI** by **11:59 PM on Friday November 5 via D2L**. There will be an in-class overview on October 19 and work session on October 21.~~

Follow the instructions in our Assignment 1 for setting up your sketchbook location or download the Processing 3.x environment if you need to do this. You probably don't need to do this. However, if your laptop caused unresolved problems with 3D Assignment 2, then use the campus Windows PCs that have access to S: networked drive. Windows computer labs on the KU campus such as the CSC Lab in Old Main 248A at the end of the CSC office hallway and in Rohrbach Library should allow you to run Processing from the networked S:\ComputerScience\processing3.5.4 subdirectory.

Save a copy of **YOUR working Assignment 2 solution** sketch from Processing as **CSC220F21Assn3_MIDI** by running File -> Save As -> CSC220F21Assn3_MIDI. You will copy SOME code from my handout file. The PORTIONS of modified code that you must copy are here https://faculty.kutztown.edu/parson/fall2022/CSC220F21Assn3_DEMO.txt (fall 2021 handout code). The **MIDI code Tab** is here https://faculty.kutztown.edu/parson/fall2022/MIDI3aDemo.txt. You must copy & insert all of this MIDI tab's code into your sketch's new MIDI tab. If your Assignment 2 code is broken, work with me in office hours via Zoom EARLY to get your bugs fixed.
UPDATE FALL 2022: My solution to last year's problem is here
https://faculty.kutztown.edu/parson/fall2022/CSC220F21Assn3_DEMO_Parson.txt
(main sketch) and here https://faculty.kutztown.edu/parson/fall2022/MIDI3aDemoParson.txt (MIDI TAB).

ADDED 10/16:

From the 10/15 grade report sheet: If you have a running sketch that lost points for a problem such as not creating your own PShape, don't bother fixing that for assignment 3. Concentrate on the MIDI requirements for assignment 3. I will not deduct the same points twice AS LONG AS YOUR SKETCH RUNS AND MEETS THE MIDI REQUIREMENTS. You may use my makeCustomPShape() from assignment 2 if you need that.

I did not make any late charges to the projects graded 10/15 because of the new Mac problem running 3D sketches. I will make sure to include a frameRate() call in setup() to avoid the problem in future handout code.

Revision to the above paragraph on 10/16: If you lost most or all of the 25% for not coding your own makeCustomPShape() according to spec, you can earn back up to 15 of those 25 points by doing it according to Assignment 2 spec for Assignment 3. This amounts to a one-day late charge to be fair to the class. Add an ALL-CAPS comment near the top after your name telling me if you make this addition so I know to look for it.

Revision 2 on 10/16: If you lost most or all of the 50% for not coding your own 3D avatar class according to spec – there were a few that made trivial changes to a few of the numbers and turning in a professor avatar -– you can earn back up to 35 of those 50 points by doing it according to Assignment 2 spec for Assignment 3. This amounts to a 1.5-day late charge to be fair to the class. Add an ALL-CAPS comment near the top after your name telling me if you make this addition so I know to look for it.

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:
   My setup() function some added arguments for constructor calls, and draw() requires a few lines of added code at the top. **Do NOT over-write your setup() or draw().** Instead, make the small adds documented with STUDENT comments in my handout setup() and draw().

   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 in MIDI tab. You do not need to add these.

```
// 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
```

   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/fall2021/ConcentricCirclesIntervals.txt. It has its own MIDI Tab linked here https://faculty.kutztown.edu/parson/fall2021/MIDI.txt.

You can experiment with PROGRAM_CHANGE (a.k.a. patches) and CONTROL_CHANGE (a.k.a. audio effects) when running **ConcentricCirclesIntervals** by typing:

> **pN** followed by Return (Enter) where N is the number of a patch to try out.
> **cN** followed by Return (Enter) where N is the number of a CONTROL to try out.
> Hit upper-case **F** to cycle between amount 0, 64, and 127 for that CONTROLler.

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

```
// STUDENT A 10%: Add CHANNEL, PROGRAM_CHANGE, and true|false for isStereo
//   to your 3D avatar constructor calls and constructor functions, similar to
//   my Professor and VectorAvatar adds described here:
//
//   Classes Professor and VectorAvatar constructors take 3 new arguments:
//   CHANNEL (0..15), PATCH (PROGRAM_CHANGE), and true|false on whether to us
//   the stereo Pan 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 panning.
  //    I gave the other little Professors the same channel and CONTROL 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.
  //    See other upper-case STUDENT comments below.

// STUDENT B 5%: Add the MIDIready variable and the "if (! MIDIready)" block of code
// at the top of my draw() function into the top of your draw() function.
boolean MIDIready  = false ;
void draw() {
  if (! MIDIready) {
    initMIDI();
    MIDIready = true ;
  }

void keyPressed() {
  // STUDENT C 15%: implement these global variable changes,
  // MAKE SURE NOT TO GO OUTSIDE VALID RANGES!!! TEST THAT.
  //  THESE CONTROLS MUST NOT CAUSE ARRAY INDICES TO GO OUT OF BOUNDS.
  //
  //  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
if (key == CODED) {
    if (keyCode == UP) {

    } else if (keyCode == DOWN) {

    } else if (keyCode == RIGHT) {

    } else if (keyCode == LEFT) {

    }
```

NOTE FOR UP & DOWN, I received the question from several students about how to apply the new volume level after you update it:

You need to loop over the 16 channels and send a ShortMessage.CONTROL_CHANGE to each with the new volume level as the fourth argument to sendMIDI(), modeled after the

```
sendMIDI(ShortMessage.CONTROL_CHANGE,
```

function calls elsewhere in the sketch. There is a loop **in the MIDI tab** that starts like this:

```
for (int c = 0 ; c < 16 ; c++) {
```

You can use that to guide sending the new volume to each channel.

ANOTHER Q&A:

- Do we need to do something similar with the **curscale**?

Nope. Just make sure **curscale** does not go outside the size of the **scales** array defined in the MIDI tab. It must stay between 0 and **scales.length-1** inclusive.

// **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
// PROGRAM_CHANGE and 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 PROGRAM_CHANGE and CONTROL_CHANGE effect to display(),
// and have display() play notes (NOTE_ON & NOTE_OFF) similar to class Professor.

We will use the one or two classes 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 **CSC220F21Assn3_MIDI.pde** in your sketch folder. Drag **CSC220F21Assn3_MIDI.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 CSC220F21Assn3_MIDI.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. 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.

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.