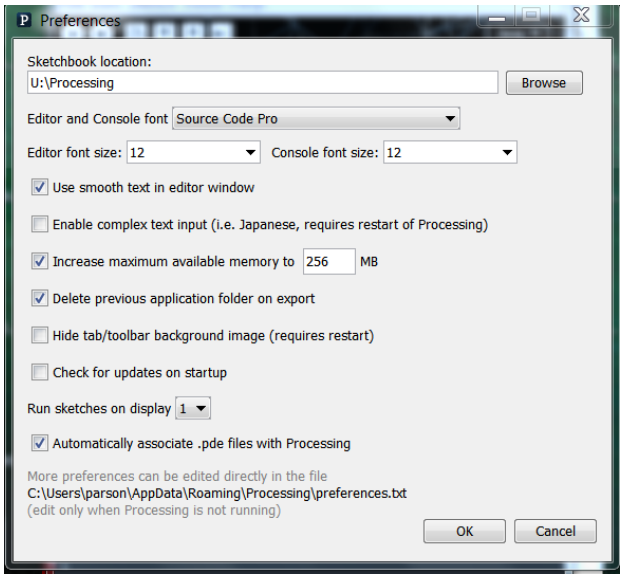
CSC 220 Object Oriented Multimedia Programming, Fall 2019

Dr. Dale E. Parson, Assignment 3, a 2D/3D visual-music MIDI virtual keyboard in Processing.
This assignment is due via **D2L Assignments** **Assignment 3 MIDI** by **11:59 PM on 8 November**.

When using Processing on the Kutztown campus Windows computers, make sure to start out **every time** by setting your Processing Preferences -> Sketchbook Location to U:\Processing. The U:\ drive is a networked drive that will save your work and make it accessible across campus. If you save it to your desktop or the lab PC you are using, you will lose your work when you log out. You must save it to the U:\ drive. If you do not have a folder called Processing under U:\, you must create one using the Windows Explorer. Processing Preferences is under the File menu on Windows.



If you will be downloading Processing 3.X and running it using an off-campus computer (do not use version 2.X for assignments), you can copy your project sketch named **CSC220F19MIDIassn3** to a flash drive on one machine, and then copy it from the flash drive to another Processing sketch folder.

You can copy and paste the text for my handout sketch into a new sketch, then save it, as a starting point. Create your **CSC220F19MIDIassn3** folder by running File -> Save As -> **CSC220F19MIDIassn3** after setting up your sketch folder. There are four .pde files in the handout code. The main one has the same name as the sketch, **CSC220F19MIDIassn3**. The other three are **Note**, **Musician**, and **CartesianPolar**. See Assignment 3 on the course page https://faculty.kutztown.edu/parson/fall2019/CSC220Fall2019.html to see how to start the project uses these files.

REQUIREMENTS (please reread this to make sure you didn't miss any when turning in):

Search for STUDENT comments (upper case) in the handout code to find these regions of code.

1. (10%) Set sketch variables **programNumber** (for instrument voice per this list[1], subtracting 1 since the Java MIDI API starts at 0, not 1) and **controlEffectNumber** for an effect[2], to numbers customized

[1] http://midi.teragonaudio.com/tutr/gm.htm
[2] http://midi.teragonaudio.com/tech/midispec.htm

for your instrument and voice. Add a comment at the top, after your name, telling me the values you selected for these two variables. "Finding your voice" requires some experimentation. Also, if your name or this comment is missing from the top of the sketch, I will deduct points. It is hard to keep track of assignment grading when your name and required documentation are missing. Try to find a **controlEffectNumber** that has some audible effect with your **programNumber** when **controlEffectData2** is set to 127. You have to change these variable values in the code. Sketch CSC220F19MIDIc (see course page) allows you to experiment with **programNumber** using LEFT and RIGHT arrows, but I have disabled **controlEffectNumber** exploration via UP and DOWN arrows because some effects silence the MIDI channel altogether.

2. (22.5%) You must rewrite function **mapXY()** in the main sketch tab to set permanent locations for Note objects in the display. Handout mapXY() assigns random locations, resulting in this unintuitive layout.

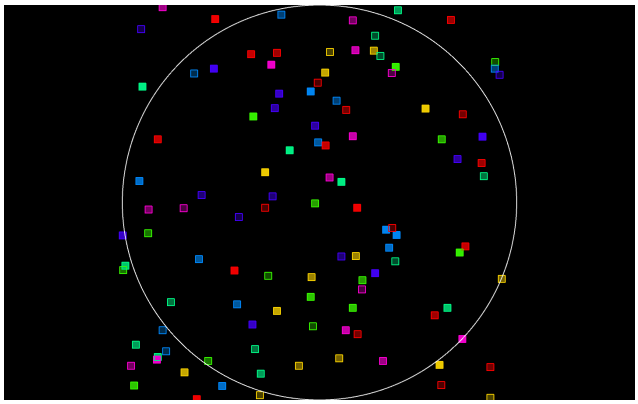


Figure 1: Handout, random layout of Note objects located by mapXY().

That outer circle (toggled using 'b') gives the base of the planetarium dome. No note may be **entirely** outside of it. Comments above mapXY() give the design requirements:

```
// STUDENT REQUIREMENTS FOR mapXY(): You must rewrite mapXY so that:
// 1. It always returns a location within the "dome", which is a circle centered
// on the sketch's 0,0,0 coordinate, which is the center of the window or projector.
// 2. Try to avoid placing two Note objects at the same location. My solution (not shared)
// accomplishes that by using parameters pitch, minpitch, and maxpitch to position
// the Notes in a circle around the center of the diplay, and uses velocityStep and
// velocityStepsTotal to determine the distance of each Note from the 0,0,0 center.
// I will give you pseudocode for my solution.
// 3. You may use an approach other than mine in requirement 2 (previous paragraph);
// if you do, try to arrange Notes in some logical, regular order by pitch & velocityStep.
// NOTE: If you use fewer than all of this function's parameters, you will get
// warnings (not errors) about unused parameter(s). Just ignore those warnings.
int [] mapXY(int pitch, int minpitch, int maxpitch, int velocityStep, int velocityStepsTotal)
```

See the handout code comments for pseudocode that explains using helper functions in the CartesianPolar tab to lay the Notes out using polar geometry, as in Figure 2. Polar geometry does not use X,Y coordinate pairs. Instead, it uses RADIUS, ANGLE coordinate pairs. A RADIUS of 0 is the center of the display, and a RADIUS of 1 is the edge of the bounding circle. An ANGLE of 0 goes from center to the right in the above figure, proceeding counterclockwise by degrees until coming full circle at TWO_PI radians (360 degrees). I supply conversion functions to go from polar coordinates to display coordinates, along with

mapXY() pseudocode in the handout. Feel free to come up with a different approach, or you can use this.

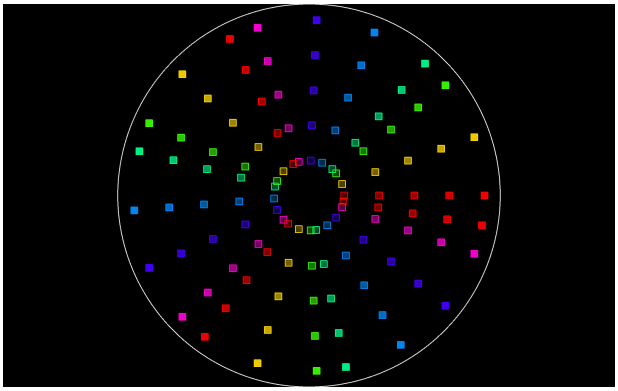


Figure 2: My solution's Polar geometry Note layout.

I have posted color copies of Figure 1 and Figure 2 on the course page.

3. (22.5%) The **Note** tab in the handout code defines class Note. Please read and understand that code, including how it works with the rest of your sketch. We will go over all handout code in class. Your first Note requirement is to replace the rect() function call in Note.display() with a different 2D shape that is at least as wide and high as mine. Look for the first STUDENT comment in the Note tab.
4. (22.5%) Your next requirement is to replace the box() function call within Note.display() with a different 3D shape to be displayed with similar size in the X,Y,Z directions when that Note is being sounded. You can try a sphere deformed using scale(X,Y,Z), or you can use the PShape function that creates a cylinder in demo script CSC220F19DemoPShapeCylinder, also linked on the course page. Or, you can use my PShape or your own custom PShape, including mine or yours from assignment 2. Test to make sure it works and has some visual appeal. Scale the Z depth by the velocity, similar to my current box() call. Louder notes must make deeper (in Z) 3D shapes than more quiet notes. See STUDENT comment in Note.
5. (22.5%) Implement function Note.setShearXSpeed(float speedInDegrees), see STUDENT comments. You will need two data fields in the class, one to store current shearX amount in degrees, and one to store this speed. Apply the shearX() function in Note.display at the appropriate place. When the speedInDegrees parameter is 0, set both the object's shearX amount and its shearX speed to 0, i.e., no shearing. Use the approach used for updating 2D rotation in degrees from earlier sketches that had both speed and location variables. You can just let the shearX amount variable continue past 360 or -360 degrees, since degrees just wrap around at that point. Here is a video of shearing at 20 seconds.

TURNING IT IN: When your work is completed, and you have re-read and satisfied project requirements (including all STUDENT instructions in code comments), you can use the Windows Explorer to find the file **CSC220F19MIDIassn3** folder; you must ZIP (standard zip format) & turn in the entire folder, since it contains multiple files. If you find you have created an error, you can drop an updated assignment 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. Finally, because your sketch uses multiple files, turn in a .zip archive of the entire CSC220F19MIDIassn3/ directory instead of the individual files. I cannot run a sketch that depends on additional files without receiving those files. I will deduct points if there are missing files.